

no page table

76



NAME: Sara Deneen  
Comp 431

Exam # 2

NUMBER: P  
11/12/2004

[1] Fill in blanks:

- The average internal fragmentation in a paging system is

~~external external fragmentation~~

- The best and fastest implementation of a page table is  
with Registers

- The advantage of using paging in memory management

1- it reduces overhead & there will not be an external fragmentation

- 2- sharing pages

- In MVT, the poorest allocation method is

worst fit

- The solution for external fragmentation in MVT is

The compaction

- If a process most of the time is busy swapping pages in & out of memory, then this is called utilization of CPU

The page fault and need a high S

- Starting execution with zero pages in memory is called

it is not in actual memory but it is initial

- The best page replacement algorithm which gives the minimum page faults is

optimal Algorithm

- In a paging system, if  $LA = u$ , page size =  $s$ , then

$$\text{page No., } p = \frac{u}{s}$$

$$\text{page offset, } d = \frac{u \% s}{s}$$

is The page table entry is full or Empty

[2] (a) Define briefly the following bit types associated with the page table.

(i) legal / illegal Bit

its a bit add to the page table to indicate if the page has a legal or not it

has add data or full page every region in the page table has a page the valid bit is set to 1. otherwise it set to 0

(ii) valid / invalid Bit

it a bit that has been add to the page table to indicate if the page is in the memory or not

(iii) protection Bit

it is a bit that add to page table to indicate if the page is read only or we can modify it or read-write page so we can modify the page

(iv) reference Bit

its a bit that add to page table to indicate if the page has been referenced (executed or not)

(b) (i) What is page fault?

The processor finds the page is not in memory

(ii) What are the steps executed by the page fault service routine?

1 to check is the memory is full or not if not it swap id not then choose an victim by some algorithm

2 - swap out to check if the victim page has been modified or not if it modified it swap it to memory and restore it if not it replace it or swap in the needed pages in this page

3 compute the physical address of this page and get instruction to make an execute for it

[3] In a computer system with paging memory management, if the page size is 8 KB,  
memory contains 128000 frames. If the maximum program can be executed in  
this machine contains 2048 pages, then compute:

- The No. of bits in the LA

$$\text{The size of program is } 2048 \text{ bytes} = 2^{11} = 11 \text{ bit} = p = 11$$

$$\text{But The page size} = 8 \text{ KB} = 8 \times 2^{10} = 2^{13} = d = 13$$

$$LA = 11 + 13 = 24$$

11	13
----	----

- The No. of bits in the PA

$$\text{The memory size contains } 128000 = 2^{17} = 17 \text{ bit}$$

$$\text{The page size} = 2^{13} = d = 13$$

$$PA = 17 + 13 = 30 \text{ bit}$$

- The physical memory size

$$\text{The physical size} = 2^{17} \times 2^{13} = 2^{30}$$

$$" 128000 \times 2^{12} =$$

$$2^{17} \times 2^{13} = 2^{30}$$

- The page table size

$$\text{Entry 17 bit need 3 byte so}$$

$$\text{The page table entry} = 2048 = 2^{11}$$

$$\text{Then The page size} = 2048 \times 3 = 6144 \text{ byte}$$

- The maximum program size can be executed

$$\text{The maximum program size} = 2048 \times 8 \text{ KB}$$

$$= 2048 \times 2^{13}$$

$$= 2^{11} \times 2^{13}$$

$$= 2^{24} \text{ byte}$$

[4] (a) An OS supports a demand paging system, Given:

page fault rate  $p$

memory access  $m$

page fault service routine time  $t$

Compute the EAT?

$$(1-p)(m) + p(t + 2t + 2m)$$

↓  
sum in only

memory address

Memory access,      sum in only      memory address

(b) Given that:

Memory access is 100 micS

Page transfer time 5 milS 5000 micu

40% of the time the page is modified

If the EAT = 1500 micS,

Compute the page fault rate  $p$ .

$$\text{EAT} = (1-p)(m) + p \left[ .4 \text{ sum in } + .4 \times \text{sum of modified pages} + \text{memory access} \right]$$

$$1500 = (1-p)(100) + p \left[ (0.4 \times 5000) + (0.4 \times 2 \times 5000) + 200 \right]$$

$$1500 = 100 - 100p + p \left\{ 2000 + 4000 + 200 \right\}$$

$$1500 = -100p + 7200$$

$$1500 = 7100p$$

$$p = \frac{1500}{7100} = .2125$$

$$\approx .23$$



*Computer Science Department*

$$\begin{array}{r} 120 \times \\ 2 \\ \hline 240 \end{array}$$

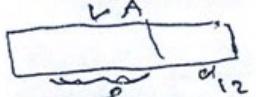
NAME : \_\_\_\_\_  
Comp 431

Exam # 2

17/05/2006

- [1] In a computer system with paging memory management, if the page size is 4 KB, memory contains 256000 frames. If the maximum program can be executed in this machine contains 8192 pages, then compute by showing your work:  $4 \times 2^{10} : 2^{12}$

- The No. of bits in the LA



$$\text{Per size, } ux_2^{10} = \frac{1}{2}^2 \\ \text{Per row} = 8192 \approx 2^{12} \\ \therefore LA = 2 \text{abit}$$

- The No. of bits in the PA

$$\text{frame num} = 286000$$

$$\approx \frac{8}{2 \times 2^{10}} = 2^{12}$$

17 bit

$$\begin{array}{r}
 8192 \\
 \times 2 \\
 \hline
 16384 \\
 + 16384 \\
 \hline
 32768 \\
 \end{array}
 \quad
 \begin{array}{r}
 256900 \\
 \times 2 \\
 \hline
 513800 \\
 + 513800 \\
 \hline
 1027600 \\
 \end{array}$$

- ### - The size physical memory

$$\underline{23600} \times \underline{2} \quad \checkmark = \underline{2}$$

- = off frameX framesize

515

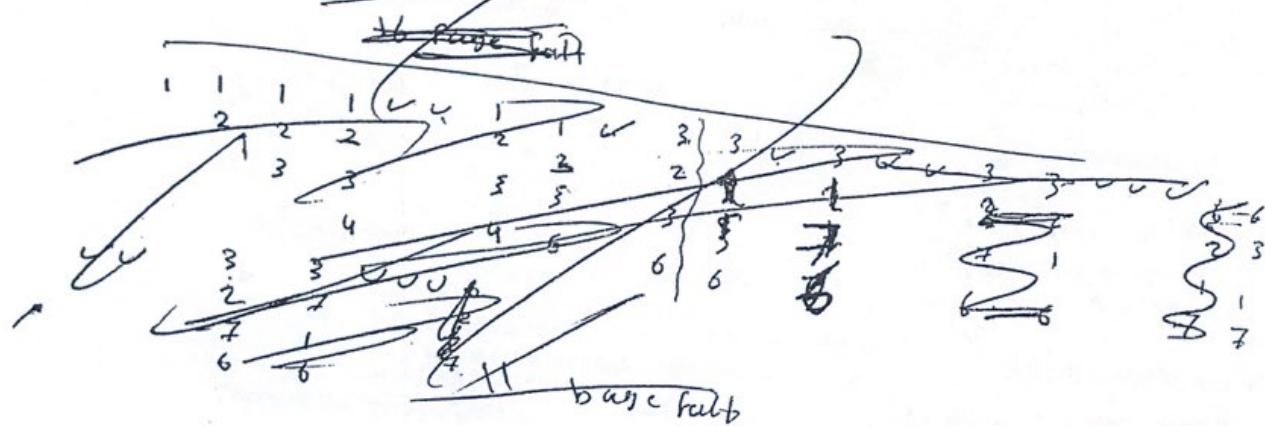
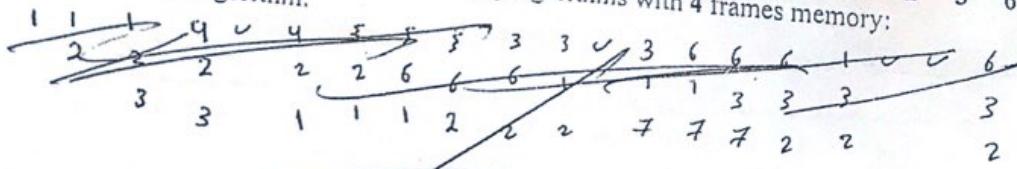
~~8387255000~~

- The page table size

12  
24

- [2] (a) In executing a certain process, we have the following page references:

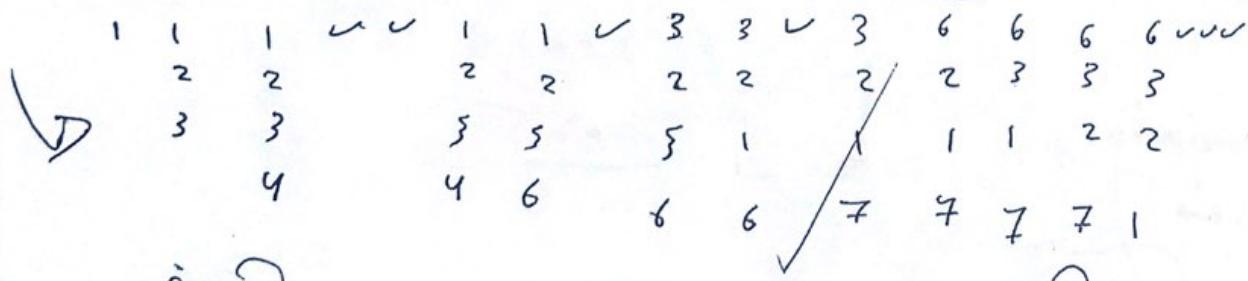
1 2 3 4 2 1 5 6 / 2 3 / 1 3 7 / 6 3 2 1 2 3 6  
 Compute the page fault rate for the following algorithms with 4 frames memory:  
 LRU replacement algorithm.



- (b) In a demand paging system where page table is kept in registers. If servicing a page fault takes 1 millisecond if a free frame is available or the page is not modified and 2 milliseconds if page is modified. If memory access is 100 microsecond, and 60% of the time the page is modified. If the page fault rate  $p = 30\%$ . Compute the EAT.

$$(1 - 0.3) * 100 + 0.3 \left( \frac{60}{100} * 2 + 1 + 100 \right)$$

→ 51 ms



→ 12 Page Fault

### [3] Fill in blanks:

- The average internal fragmentation ~~is~~ <sup>in</sup> ~~paging system~~ ~~1/2 pag~~
- The advantage of using paging memory management ~~Page Sharing~~
- In MVT, the best allocation algorithm is ~~best fit~~, and the worst is ~~worst fit~~
- If a process most of the time is busy swapping pages in & out of memory, then this is called ~~thrashing~~
- Starting execution with zero pages in memory is called ~~zero demand paging~~
- In a paging system, if  $LA = u$ ,  $page\ size = s$ , then
  - page No.,  $p = \frac{u}{s}$
  - page offset,  $d = s$
- The bit that indicates if the page is in memory ~~valid linkage bit~~
- The bit that indicates if the page is changed ~~dirty bit~~
- The bit that indicates if the page read or written ~~reference bit~~
- In a demand paging system where the degree of multiprogramming is 5 and the status of the system is measured, and it is as follows:

CPU utilization 15%, Disk utilization 95%

Can we increase the degree of multiprogramming to improve the performance of the system,

why? Yes ~~because~~ <sup>Disk usage 95% no swapped in</sup> ~~need more~~  
~~or one~~  $\Rightarrow$  there exist frame in memory  
 $\Rightarrow$  we can increase the no of task in memory

$\frac{316}{52}$        $\frac{22}{2}$        $\frac{16}{4}$   
 $\frac{22}{2}$        $\frac{16}{4}$   
 $\frac{32}{4}$        $\frac{16}{4}$   
 $\frac{64}{8}$        $\frac{16}{4}$   
 $\frac{128}{16}$        $\frac{16}{4}$   
 $\frac{256}{32}$        $\frac{16}{4}$   
 $\frac{512}{64}$        $\frac{16}{4}$   
 $\frac{1024}{128}$        $\frac{16}{4}$   
 $\frac{2048}{256}$        $\frac{16}{4}$   
 $\frac{4096}{512}$        $\frac{16}{4}$   
 $\frac{8192}{1024}$        $\frac{16}{4}$   
 $\frac{16384}{2048}$        $\frac{16}{4}$   
 $\frac{32768}{4096}$        $\frac{16}{4}$   
 $\frac{65536}{8192}$        $\frac{16}{4}$   
 $\frac{131072}{16384}$        $\frac{16}{4}$   
 $\frac{262144}{32768}$        $\frac{16}{4}$   
 $\frac{524288}{65536}$        $\frac{16}{4}$   
 $\frac{1048576}{131072}$        $\frac{16}{4}$   
 $\frac{2097152}{262144}$        $\frac{16}{4}$   
 $\frac{4194304}{524288}$        $\frac{16}{4}$   
 $\frac{8388608}{1048576}$        $\frac{16}{4}$   
 $\frac{16777216}{2097152}$        $\frac{16}{4}$   
 $\frac{33554432}{4194304}$        $\frac{16}{4}$   
 $\frac{67108864}{8388608}$        $\frac{16}{4}$   
 $\frac{134217728}{16777216}$        $\frac{16}{4}$   
 $\frac{268435456}{33554432}$        $\frac{16}{4}$   
 $\frac{536870912}{67108864}$        $\frac{16}{4}$   
 $\frac{1073741824}{134217728}$        $\frac{16}{4}$   
 $\frac{2147483648}{268435456}$        $\frac{16}{4}$   
 $\frac{4294967296}{536870912}$        $\frac{16}{4}$   
 $\frac{8589934592}{1073741824}$        $\frac{16}{4}$   
 $\frac{17179869184}{2147483648}$        $\frac{16}{4}$   
 $\frac{34359738368}{4294967296}$        $\frac{16}{4}$   
 $\frac{68719476736}{8589934592}$        $\frac{16}{4}$   
 $\frac{137438953472}{17179869184}$        $\frac{16}{4}$   
 $\frac{274877906944}{34359738368}$        $\frac{16}{4}$   
 $\frac{549755813888}{68719476736}$        $\frac{16}{4}$   
 $\frac{1099511627776}{137438953472}$        $\frac{16}{4}$   
 $\frac{219902325552}{274877906944}$        $\frac{16}{4}$   
 $\frac{439804651104}{549755813888}$        $\frac{16}{4}$   
 $\frac{879609302208}{1099511627776}$        $\frac{16}{4}$   
 $\frac{1759218604416}{219902325552}$        $\frac{16}{4}$   
 $\frac{3518437208832}{439804651104}$        $\frac{16}{4}$   
 $\frac{7036874417664}{879609302208}$        $\frac{16}{4}$   
 $\frac{14073748835328}{1759218604416}$        $\frac{16}{4}$   
 $\frac{28147497670656}{3518437208832}$        $\frac{16}{4}$   
 $\frac{56294995341312}{7036874417664}$        $\frac{16}{4}$   
 $\frac{112589990682624}{14073748835328}$        $\frac{16}{4}$   
 $\frac{225179981365248}{28147497670656}$        $\frac{16}{4}$   
 $\frac{450359962730496}{56294995341312}$        $\frac{16}{4}$   
 $\frac{900719925460992}{112589990682624}$        $\frac{16}{4}$   
 $\frac{1801439850921984}{225179981365248}$        $\frac{16}{4}$   
 $\frac{3602879701843968}{450359962730496}$        $\frac{16}{4}$   
 $\frac{7205759403687936}{900719925460992}$        $\frac{16}{4}$   
 $\frac{14411518807375872}{1801439850921984}$        $\frac{16}{4}$   
 $\frac{28823037614751744}{3602879701843968}$        $\frac{16}{4}$   
 $\frac{57646075229503488}{7205759403687936}$        $\frac{16}{4}$   
 $\frac{115292150459006976}{14411518807375872}$        $\frac{16}{4}$   
 $\frac{230584300918013952}{28823037614751744}$        $\frac{16}{4}$   
 $\frac{461168601836027904}{57646075229503488}$        $\frac{16}{4}$   
 $\frac{922337203672055808}{115292150459006976}$        $\frac{16}{4}$   
 $\frac{1844674407344111616}{230584300918013952}$        $\frac{16}{4}$   
 $\frac{3689348814688223232}{561168601836027904}$        $\frac{16}{4}$   
 $\frac{7378697629376446464}{1122337203672055808}$        $\frac{16}{4}$   
 $\frac{14757395258752892928}{2244674407344111616}$        $\frac{16}{4}$   
 $\frac{29514790517505785856}{4489348814688223232}$        $\frac{16}{4}$   
 $\frac{59029581035011571712}{8978697629376446464}$        $\frac{16}{4}$   
 $\frac{118059162070223543424}{17957395258752892928}$        $\frac{16}{4}$   
 $\frac{236118324140447086848}{35914790517505785856}$        $\frac{16}{4}$   
 $\frac{472236648280894173696}{71829581035011571712}$        $\frac{16}{4}$   
 $\frac{944473296561788347392}{143659162070223543424}$        $\frac{16}{4}$   
 $\frac{1888946593123576694784}{287318324140447086848}$        $\frac{16}{4}$   
 $\frac{3777893186247153389568}{574636648280894173696}$        $\frac{16}{4}$   
 $\frac{7555786372494306779136}{1149273293123576694784}$        $\frac{16}{4}$   
 $\frac{15111572744886513588272}{22985465862494306779136}$        $\frac{16}{4}$   
 $\frac{30223145489773027176544}{45970931724886513588272}$        $\frac{16}{4}$   
 $\frac{60446290979546054353088}{91941863449773027176544}$        $\frac{16}{4}$   
 $\frac{120892581959092108706176}{18388372689546054353088}$        $\frac{16}{4}$   
 $\frac{241785163918184217412352}{36776745379092108706176}$        $\frac{16}{4}$   
 $\frac{483570327836368434824704}{735534907581842108706176}$        $\frac{16}{4}$   
 $\frac{967140655672736869649408}{147106981516368434824704}$        $\frac{16}{4}$   
 $\frac{193428131134547373929816}{294213963032736869649408}$        $\frac{16}{4}$   
 $\frac{386856262269094747859632}{58842792606547373929816}$        $\frac{16}{4}$   
 $\frac{773712524538189495719264}{117685585213094747859632}$        $\frac{16}{4}$   
 $\frac{1547425049076378991438528}{235371170426189495719264}$        $\frac{16}{4}$   
 $\frac{3094850098152757982877056}{470742340852378991438528}$        $\frac{16}{4}$   
 $\frac{6189700196305515965754112}{941484681704757982877056}$        $\frac{16}{4}$   
 $\frac{1237940039261103193150824}{1883017363409515965754112}$        $\frac{16}{4}$   
 $\frac{2475880078522206386301648}{376603472681903193150824}$        $\frac{16}{4}$   
 $\frac{4951760157044412772603296}{753206945363806386301648}$        $\frac{16}{4}$   
 $\frac{9903520314088825545206592}{1506413890727612772603296}$        $\frac{16}{4}$   
 $\frac{19807040628177651090413184}{3012827781455225545206592}$        $\frac{16}{4}$   
 $\frac{39614081256355302180826368}{6025655562910451090413184}$        $\frac{16}{4}$   
 $\frac{79228162512710604361652736}{12051311125820902180826368}$        $\frac{16}{4}$   
 $\frac{15845632522542120872325552}{24102622251641804361652736}$        $\frac{16}{4}$   
 $\frac{31691265045084241744651104}{4820524450328360872325552}$        $\frac{16}{4}$   
 $\frac{63382530090168483489302208}{9641048900656721744651104}$        $\frac{16}{4}$   
 $\frac{126765060180336966978604416}{19282097801313443489302208}$        $\frac{16}{4}$   
 $\frac{253530120360673933957208832}{38564195602626886978604416}$        $\frac{16}{4}$   
 $\frac{507060240721347867914417664}{77128391205253773957208832}$        $\frac{16}{4}$   
 $\frac{101412048144269573582835328}{154256782410507547914417664}$        $\frac{16}{4}$   
 $\frac{202824096288539147165670656}{30851356482101509582835328}$        $\frac{16}{4}$   
 $\frac{405648192577078294331341312}{61702712964203019165670656}$        $\frac{16}{4}$   
 $\frac{811296385154156588662682624}{123405425928406038331341312}$        $\frac{16}{4}$   
 $\frac{1622592770308313177325365248}{246810851856812076662682624}$        $\frac{16}{4}$   
 $\frac{3245185540616626354650730496}{493621703713624153325365248}$        $\frac{16}{4}$   
 $\frac{6490371081233252709301460992}{987243407427248306650730496}$        $\frac{16}{4}$   
 $\frac{12980742162466505418602921984}{1974486814854496613301460992}$        $\frac{16}{4}$   
 $\frac{25961484324933010837205843968}{3948973629708993226602921984}$        $\frac{16}{4}$   
 $\frac{51922968649866021674411687936}{7897947259417986453205843968}$        $\frac{16}{4}$   
 $\frac{103845937299732043348823375872}{15795894518835972906411687936}$        $\frac{16}{4}$   
 $\frac{207691874599464086697646751744}{315917890376719458128233751744}$        $\frac{16}{4}$   
 $\frac{415383749198928173395293503488}{631835780753438916256467503488}$        $\frac{16}{4}$   
 $\frac{830767498397856346790587006976}{1263671561506877832512935006976}$        $\frac{16}{4}$   
 $\frac{1661534977957712693871174013952}{2527343123013755665025870013952}$        $\frac{16}{4}$   
 $\frac{3323069955915425387742348027856}{5054686246027511330051740027856}$        $\frac{16}{4}$   
 $\frac{6646139911830850775484696055712}{10109372492055022660103480055712}$        $\frac{16}{4}$   
 $\frac{13292279823661701550969392111424}{20218744984110045320206960055712}$        $\frac{16}{4}$   
 $\frac{26584559647323403001938784222848}{40437489968220090640413920055712}$        $\frac{16}{4}$   
 $\frac{53169119294646806003877568445696}{80874979936440181280827840055712}$        $\frac{16}{4}$   
 $\frac{10633823858929361200775136881192}{161749959872880362561655680055712}$        $\frac{16}{4}$   
 $\frac{21267647717858722401550273616384}{323499919745760725123211360055712}$        $\frac{16}{4}$   
 $\frac{42535295435717444803050547232768}{646999839491521450246422720055712}$        $\frac{16}{4}$   
 $\frac{85070590871434889606101094465536}{1293999678983042900492845440055712}$        $\frac{16}{4}$   
 $\frac{17014118174286977921220218893112}{2587999357966085800985690880055712}$        $\frac{16}{4}$   
 $\frac{34028236348573955842440437786224}{5175998715932171601971381760055712}$        $\frac{16}{4}$   
 $\frac{68056472697147911684880875572448}{10351997431864343203942763520055712}$        $\frac{16}{4}$   
 $\frac{13611294539429582336976175114496}{20703994863728686407885527040055712}$        $\frac{16}{4}$   
 $\frac{27222589078859164673952350228992}{41407989727457372815771054080055712}$        $\frac{16}{4}$   
 $\frac{54445178157718329347904700457984}{82815979454914745631542108160055712}$        $\frac{16}{4}$   
 $\frac{108890356315436658695809400915968}{165631958909829491263084216320055712}$        $\frac{16}{4}$   
 $\frac{217780712630873317391618801831936}{331263917819658982526168432640055712}$        $\frac{16}{4}$   
 $\frac{435561425261746634783237603663872}{662527835639317965052336865280055712}$        $\frac{16}{4}$   
 $\frac{871122850523493269566475207327744}{1325055671278635930104673735560055712}$        $\frac{16}{4}$   
 $\frac{1742245701046986539132904014655488}{2650111342557271860209347471120055712}$        $\frac{16}{4}$   
 $\frac{348449140209397307826580802931096}{5300222685114543720418694942240055712}$        $\frac{16}{4}$   
 $\frac{696898280418794615653161605862192}{1060044537022906744083738988480055712}$        $\frac{16}{4}$   
 $\frac{1393796560837589231306323211724384}{2120089074045813488167477976960055712}$        $\frac{16}{4}$   
 $\frac{2787593121675178462612646423448768}{4240178148091626976334955953920055712}$        $\frac{16}{4}$   
 $\frac{5575186243350356925225292846897536}{8480356296183253952669911907840055712}$        $\frac{16}{4}$   
 $\frac{1115037248670171385045058569379512}{16960712592366507905339823815680055712}$        $\frac{16}{4}$   
 $\frac{2230074497340342770085117138759024}{33921425184733015810679647631360055712}$        $\frac{16}{4}$   
 $\frac{4460148994680685540160234277518048}{67842850369466031621359295262720055712}$        $\frac{16}{4}$   
 $\frac{8920297989361371080320468555036096}{135685700738932063242718590525440055712}$        $\frac{16}{4}$   
 $\frac{17840595978722742160640937110072192}{271371401477864126485437181050880055712}$        $\frac{16}{4}$   
 $\frac{35681191957445484321281874220144384}{542742802955728252970874362101760055712}$        $\frac{16}{4}$   
 $\frac{71362383914890968642563748440288768}{1085485605911456505941748724203520055712}$        $\frac{16}{4}$   
 $\frac{142724767829781937285127496880577536}{2170971211822913011883497448407040055712}$        $\frac{16}{4}$   
 $\frac{285449535659563874570254993761155072}{4341942423645826023766994896814080055712}$        $\frac{16}{4}$   
 $\frac{570899071319127749140509987522310144}{8683884847291652047533989793628160055712}$        $\frac{16}{4}$   
 $\frac{1141798142638255492881019975044620288}{17367769694583304095067979587256320055712}$        $\frac{16}{4}$   
 $\frac{2283596285276510985762039950089240576}{34735539389166608190135959174512640055712}$        $\frac{16}{4}$   
 $\frac{4567192570553021971524079900178481152}{69471078778333216380271918349025280055712}$        $\frac{16}{4}$   
 $\frac{9134385141106043943048159800356962304}{138942157556666432760543836698050560055712}$        $\frac{16}{4}$   
 $\frac{18268770282212087886096319600713924608}{277884315113332865521087673396101120055712}$        $\frac{16}{4}$   
 $\frac{36537540564424175772192639201427849216}{555768630226665731042175346792202240055712}$        $\frac{16}{4}$   
 $\frac{73075081128848351544385278402855798432}{111153726045333146208435069358440480055712}$        $\frac{16}{4}$   
 $\frac{14615016225769670308877055680571559664}{222307452085666292416870138716880960055712}$        $\frac{16}{4}$   
 $\frac{29230032451539340617754111361143119328}{444614904171332584833740277433761920055712}$        $\frac{16}{4}$   
 $\frac{58460064903078681235508222722286238656}{889229808342665169667480554867523840055712}$        $\frac{16}{4}$   
 $\frac{11692012906015736247101644544457247712}{1778459616685330339334961109735047680055712}$        $\frac{16}{4}$   
 $\frac{23384025812031472494203289088914495424}{3556919233370660678669922219470085360055712}$        $\frac{16}{4}$   
 $\frac{4676805162406294498840657817782898848}{711383846674132135733984443894016720055712}$

80  
100NAME: Anas Salim  
Comp 431

Exam #2

NUMBER: 102 3032  
17/12/2005

[1] (a) Select the correct answer from the list:

Parity Bit, v/i Bit, R/W Bit, Dirty Bit, Reference Bit, Flag Bit, legal/illegal Bit

R/W Bit indicates if the page is a read only bit pagelegal/illegal bit indicates if the page in the logical address spacev/i Bit indicates if the page in memoryDirty Bit indicates if the page is changedReference Bit indicates if the page is being read or written⇒ (b) In a demand paging system, assume the system status is as shown below with a very poor performance system. Briefly explain what is happening, and what do you suggest to improve it:

CPU utilization is 10% and paging disk utilization is 05%.

because the disk utilization is small so we assigned that no swapping happened which may be the # of framing is small and the degree of multiprogramming is small to improve the system; Increasing degree of multiprogramming.

(c) Explain briefly the effect of selecting large and small page size

⇒ if the page ~~table~~ large then:-  
 the external fragmentation is large  
 we need more space to save it.  
 - the internal fragmentation is large (disadvantage)  
 - the page table is small size (advantage).

⇒ if the page ~~table~~ small then:-  
 the space that we needed is small and  
 - the internal fragmentation small (advantage)  
 - the page table size is large (disadvantage).

[2] (a) In a demand paging system, Given that:

Memory access is 10 milS

Page transfer time 100 milS

30% of the time the page is modified

Page fault rate is 20%

Compute the EAT  $\rightarrow$  page fault  $\rightarrow$  page fault.

$$EAT = \{ 0.80(10) \} + 20\% (10 + \frac{10 * 30}{100})$$

$$= \frac{80}{100} + \frac{2}{10} (10 + 3)$$

$$= 8 + 2.6 \approx 10.6 \text{ mils.}$$

$$\begin{aligned} EAT &= (1 - 20) * 10 \\ &+ \frac{20}{100} (\frac{30}{100} * 1000 + 100 + 10) \end{aligned}$$

(b) In a demand paging system with a three-levels page table stored in memory and a set of associative registers. Given:

memory access = 5 milS

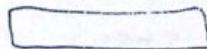
hit ratio = 95%.

Compute the EAT ignoring the lookup time in the associative registers.

$$\begin{aligned} EAT &= (\frac{95}{100} * 5) + (0.05 * 4 * 5) \\ &= 4.75 + 1 = 5.75 \text{ mils.} \end{aligned}$$

$$95\% + 0.05(4 * 5)$$

$$PA = 28 \text{ bits}$$



3 bits  
25 bits

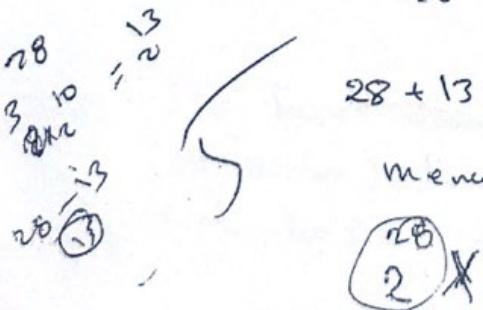
(c) If the physical address contains 28 bits with page size is 8KB, compute memory size.

$$\text{page size} = 8 \text{ KB} = 2^3 * 2^{10} = 2^{13}$$

$$\text{so } d = 13 \text{ bit}$$

$$28 + 13 = 41 \text{ bit}$$

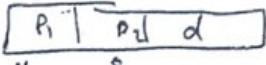
$$\text{memory size} = 2^{41} \text{ byte.}$$



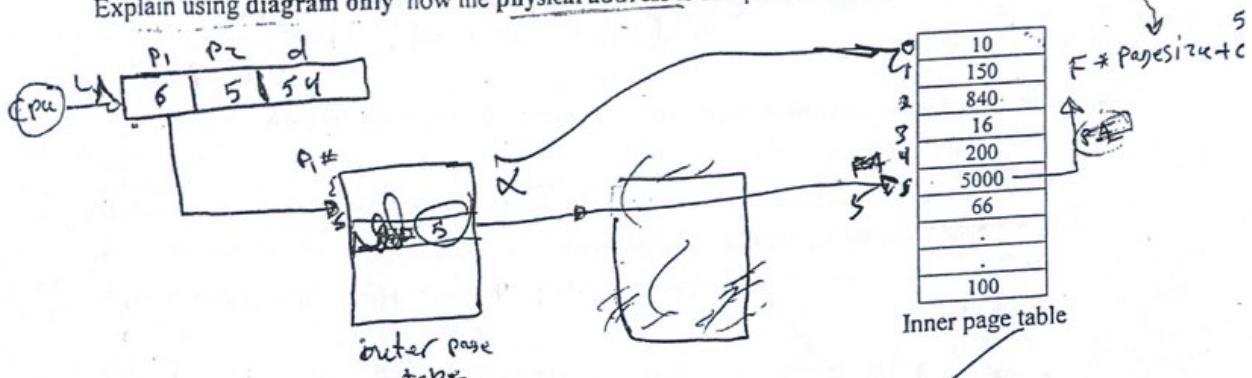
$$\begin{array}{r} 28 \\ 2 \quad 2 \\ \hline 13 \end{array} \times 2^3$$

$$\begin{array}{r} 12 \\ 2 \quad 2 \\ \hline 15 \end{array} \times 2^3$$

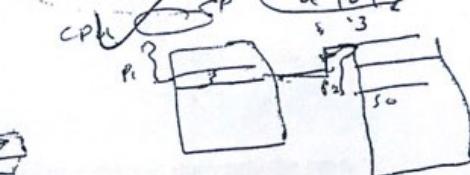
$$\begin{array}{r} 28 \\ 2 \quad 2 \\ \hline 15 \end{array} \times 2^3$$



[3] (a) In a certain computer, the virtual address is 24 bits with page size is 4096 bytes. The page table is implemented using two-levels with 4 bits for the outer page table and 8 bits for the inner. Given the virtual address in binary: 0110100000101100000011101101, Explain using diagram only how the physical address is computed from the virtual address.



$$PA = \text{base} * 4096 + \text{offset}$$



(b) For the binary semaphore S defined below.

`int S; // S is a binary semaphore`

    (i) Define the wait and signal instruction(operation) on S.

wait(S)  
 { while ( $S \leq 0$ )  
     & do nothing  
      $S = S - 1$

signal(S)  
 {  
      $S = S + 1$

wait(S)  
 do  
 { while  $S \leq 0$   
     over  
      $S = S - 1$

    if necessary  
     atomic  
      $S = S + 1$

(ii) As a condition, the wait and signal operations above must be executed atomically. Show what will happen if they don't. Explain with an example.

in wait and signal     $S = S - 1$ ; and  $S = S + 1$ ;

its critical section. if it execution not atomically  
 the sum consistence happen

the int s : is shared in the process in data.

if it executed parallel the as example..

wait(S) do  $S_0: register1 = S$ ;  
 $S_1: register1 = register1 - 1$ ;  
 $S_2: S = \text{return } S$ ;

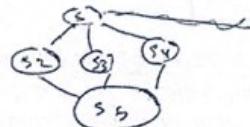
signal(S)  
 $S_0: register2 =$   
 $S_1: register2 + 1$ ;  
 $S_2: return S$ ;

if wats do  $S_0$  & the signal do  $S_0$  and  $S_1$ ;

and wats do  $S_1$ ; that can be error; L...A or!

[4] Given the statements:

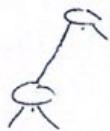
S1: int x = 10, y=5, z, c=0;  
S2: c += x;      c = c + x  
S3: y \*= 2;      y = y \* 2  
S4: cin >> z;    cin >> z  
S5: cout << x << y << z << c;



c = c + x  
y = y \* 2  
cin >> z  
cout << c

(a) Rewrite the statements above using the Dijkstra notation parbegin and parend:

S1 ;  
Parbegin  
S2 ;  
S3 ;  
S4 ;  
Parenel  
S5 ;



(b) Rewrite the statements above using fork and join structure.

S1 ;  
Fork L1:  
S2 ;

S1  
Fork L1  
Fork L2  
S2 ;

count = 3 ;

S1 ;  
Fork L1;

S2 ;  
Go to L3

L1 : Fork L2  
S3  
Go to L3

L2 : S4 ; Join count  
L3 : while (count > 0)  
do nothing  
Join S2 / S3, S4 ;

S5 ;



BIRZEIT UNIVERSITY  
College of Engineering and Technology  
Computer Science Department

Exam out of 70  
Each question  
10 points

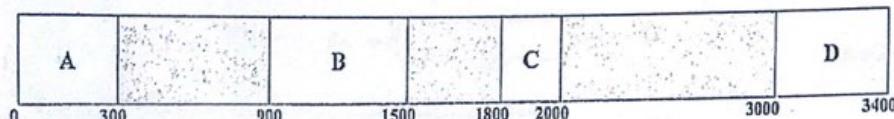
Name .....  
Comp431

Number .....  
Midterm Exam

Section 10-11 1-2  
Fall 14/15

[1] In Dynamic(variable) regions memory management (MVT), the memory status looks like:

(10)



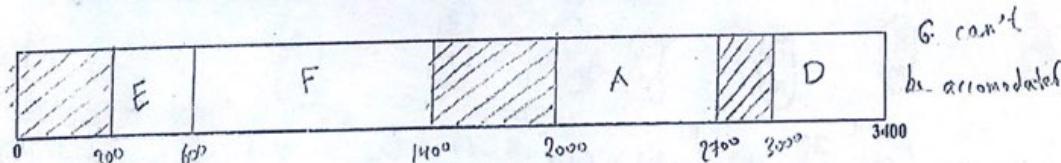
Given the following memory requests in the given order:

- Process E starts and requests 300 memory units.
- Process A requests 400 more memory units.
- Process B exits.
- Process F starts and requests 800 memory units.
- Process C exits.
- Process G starts and requests 900 memory units.

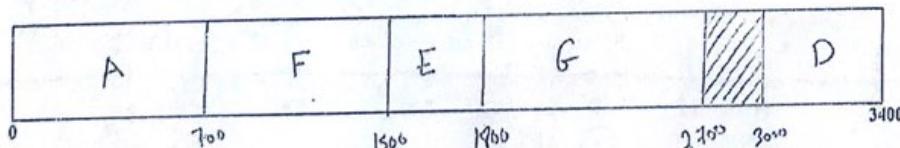
NOTE:

- For a growing process, if the current state cannot accommodate a request, reallocate the process.
- If the request can't be accommodated, just state that.

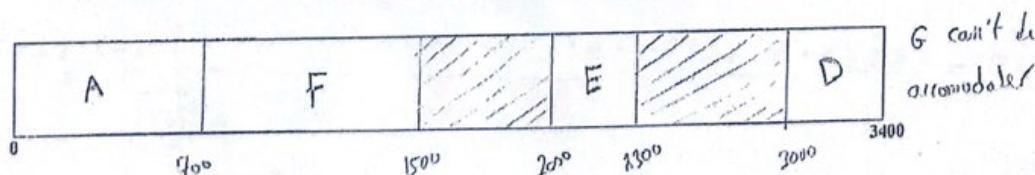
(a) Describe the contents of memory after each request using first fit



(b) Describe the contents of memory after each request using best fit.



(c) Describe the contents of memory after each request using worst fit.



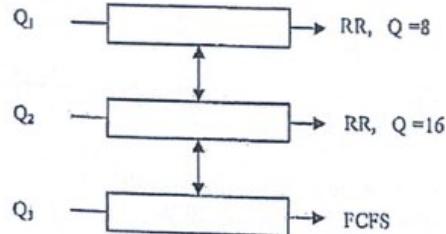
Q [3] Assume the ready queue looks like:

Process	CPU Burst	Arrival Time
P <sub>1</sub>	20' 64	00
P <sub>2</sub>	20' 14' 32	06
P <sub>3</sub>	8' 0	11
P <sub>4</sub>	10' 0	20

70 for 16<sup>th</sup>  
28 for 0  
6 0  
10 20

Given a multilevel Feedback queues system which looks as:

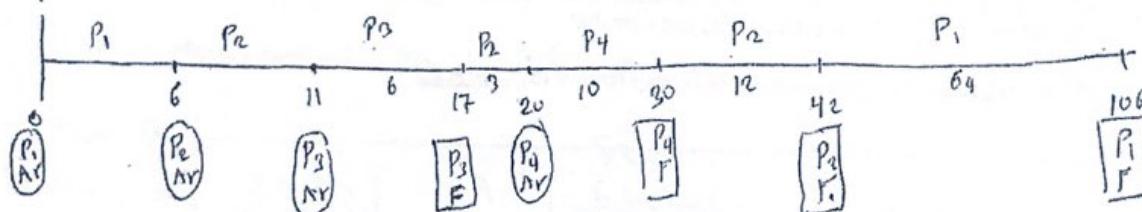
Q1 is Entry Queue



Compare the Average Waiting Time for the scheduling algorithms:

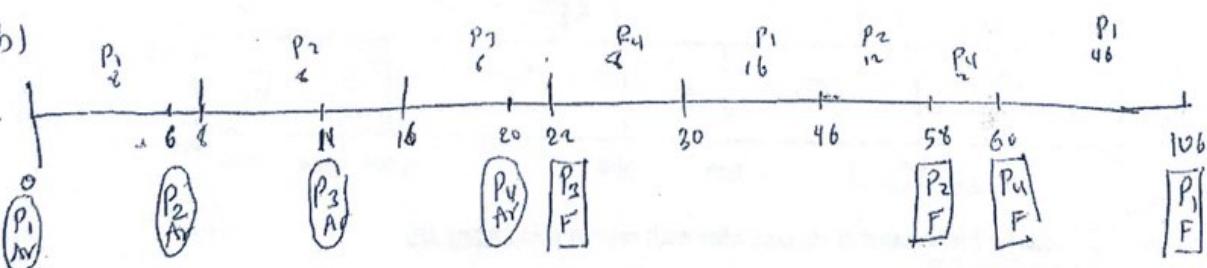
- (a) SJF with preemption (SRTF) and
- (b) Multilevel Feedback Queues

Q (a)



$$AWT = \frac{(106 - 0) + (42 - 6 - 20) + (17 - 11 - 6) + (30 - 20 - 10)}{4} = \frac{36 + 16 + 0 + 0}{4} = \boxed{\frac{52}{4}} = \boxed{13}$$

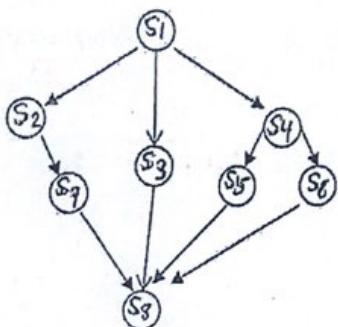
Q (b)



$$AWT = \frac{(106 - 0 - 70) + (56 - 6 - 20) + (22 - 11 - 6) + (60 - 20 - 10)}{4} = \frac{36 + 32 + 5 + 30}{4} = \boxed{\frac{103}{4}}$$

It shows that: SJTF is better algorithm for this case since it gives less AWT.

Q [6] Given the precedence graph:



Q (a) Write an equivalent code using Fork & Join

count = 4

S1

Fork L1

Fork L2

S3

goto L4

L2: S4

Fork L3

S6

goto L4

L3: S5

goto L4

L1: S2

S7

L4: join count

S8

Q (b) Write an equivalent code using parbegin & paren

Begin

S1;

Parbegin

Begin

S2;

S7;

END

[ S3;

Begin

S4;

Parbegin

S5;

S6;

Parend

END

Parend

S8;

END.

(a)

[7] (a) Is the following system of 4 processes with 2 resource types deadlocked? Show your work.

Current allocation matrix

	R1	R2
P1	1	3
P2	4	1
P3	1	2
P4	2	0

Current request matrix

	R1	R2
P1	1	2
P2	4	3
P3	1	7
P4	5	1

available vector

R1	R2
1	4
2	7
3	9

lit search for a safe sequence

(b)  $\langle P_1, P_3 \rangle$ , No other process will be satisfied

$\therefore$  There is no safe sequence

$\therefore$  may be deadlock

(b) If the available vector is as shown, is the system deadlocked? Show your work.

(b) lit search for a safe sequence

$\langle P_1, P_3, P_2, P_4 \rangle$

There is a safe sequence

No deadlock

2	4
3	7
4	9
8	10
10	10