BIRZEIT UNIVERSITY
DEPARTMENT OF COMPUTER SYSTEM ENGINEERING

ENCS339: Operating Systems — Second Semester 2010/2011

Mr. Abdel Salam Sayyad — **Second Exam** — Date: 27/4/2011

Student Name: **Typical Solutions** **Student Number: 0000000**

Question 1: (25 marks)

The Sleeping-Barber Problem: A barbershop consists of a waiting room with n chairs and a barber room with one barber chair. If there are no customers to be served, the barber goes to sleep. If a customer enters the barbershop and all chairs are occupied, then the customer leaves the shop. If the barber is busy but chairs are available, then the customer sits in one of the free chairs. If the barber is asleep, the customer wakes up the barber. Write a program to coordinate the barber and the customers (i.e. write the barber process and the customer process).

Solution:

Need to define the following variables with the indicated initial values:

```
Semaphore Customers = 0 //initially there are no customers
Semaphore Barber (mutex) = 0 //initially the barber is asleep
Semaphore accessSeats (mutex) = 1 //aquire this lock before changing
                              // the Number of Free Seats
int NumberOfFreeSeats = n // initially all seats are free
```

The barber process:

```
while(true) {
   wait(Customers);      //tries to acquire a customer
                         //if none is available he goes to sleep
   wait(accessSeats);    //the barber is now awake
                         // acquire the lock on free seats
   NumberOfFreeSeats++;  //one seat gets free
   signal(Barber);       //the barber is ready to cut hair
   signal(accessSeats);  //release the lock on free seats
   //here the barber is cutting hair
 }
```

Continued Next Page

More Space for Question 1

The customer process:

```
while(true) {
   wait(accessSeats);   // acquire the lock on free seats
   if ( NumberOfFreeSeats > 0 ) { //if there are any free seats
     NumberOfFreeSeats--; //sitting down on a seat
     signal(Customers); //notify the barber that there's a customer
     signal(accessSeats); // release the lock on free seats
     wait(Barber);      //now it's this customer's turn,
                        //but wait if the barber is busy
     //here the customer is having his hair cut
   } else {             //there are no free seats - go home
     signal(accessSeats); //release the lock on the seats
   }
 }
```

<u>Question 2</u>: (25 marks)

Consider the following snapshot of a system:

	Allocation	Max	Available
	A B C D	A B C D	A B C D
P0	0 0 1 2	0 0 1 2	1 5 2 0
P1	1 0 0 0	1 7 5 0	
P2	1 3 5 4	2 3 5 6	
P3	0 6 3 2	0 6 5 2	
P4	0 0 1 4	0 6 5 6	

Answer the following questions using the banker's algorithm:

a) What is the content of the Need matrix?
b) Is the system in a safe state?
c) If a request from process P1 arrives for (0,4,2,0), can the request be granted immediately?

<u>Solution:</u>

a) Need = Max – Available = { 0 0 0 0 , 0 7 5 0 , 1 0 0 2 , 0 0 2 0 , 0 6 4 2 }

b) Banker's algorithm:

Initially: Work = Available = { 1 5 2 0 }, finish = { *false false false false false* }

Iteration 1: for P0, Need < Work, Work = { 1 5 2 0 } + { 0 0 1 2 } = { 1 5 3 2 }, finish[0] = *true*

Iteration 2: for P2, Need < Work, Work = { 1 5 3 2 } + { 1 3 5 4 } = { 2 8 8 6 }, finish[2] = *true*

Iteration 3: for P3, Need < Work, Work = { 2 8 8 6 } + { 0 6 3 2 } = { 2 14 11 8 }, finish[3] = *true*

Iteration 4: for P4, Need < Work, Work = { 2 14 11 8 } + { 0 0 1 4 } = { 2 14 12 12 }, finish[4] = *true*

Iteration 5: for P1, Need < Work, Work = { 2 14 12 12 } + { 1 0 0 0 } = { 3 14 12 12 }, finish[1] = *true*

Therefore, the system is in a safe state, and a safe sequence is P0, P2, P3, P4, P1.

<u>Continued Next Page:</u>

More Space for Question 2

c) First we notice that the new request { 0 4 2 0 } is less than available resources { 1 5 2 0 }, and is also less than $Need_1$ { 0 7 5 0 }, and so we proceed to pretend that we granted the request and see if the new state is safe:

The new state:

	Allocation	Max	Available
	A B C D	A B C D	A B C D
P0	0 0 1 2	0 0 1 2	1 1 0 0
P1	1 4 2 0	1 7 5 0	
P2	1 3 5 4	2 3 5 6	
P3	0 6 3 2	0 6 5 2	
P4	0 0 1 4	0 6 5 6	

Need = Max – Available = { 0 0 0 0 , 0 3 3 0 , 1 0 0 2 , 0 0 2 0 , 0 6 4 2 }

Banker's algorithm:

Initially: Work = Available = { 1 1 0 0 }, finish = { *false false false false false* }

Iteration 1: for P0, $Need_0$ < Work, Work = { 1 1 0 0 } + { 0 0 1 2 } = { 1 1 1 2 }, finish[0] = *true*

Iteration 2: for P2, $Need_2$ < Work, Work = { 1 1 1 2 } + { 1 3 5 4 } = { 2 4 6 6 }, finish[2] = *true*

Iteration 3: for P3, $Need_3$ < Work, Work = { 2 4 6 6 } + { 0 6 3 2 } = { 2 10 9 8 }, finish[3] = *true*

Iteration 4: for P4, $Need_4$ < Work, Work = { 2 10 9 8 } + { 0 0 1 4 } = { 2 10 10 12 }, finish[4] = *true*

Iteration 5: for P1, $Need_1$ < Work, Work = { 2 10 10 12 } + { 1 4 2 0 } = { 3 14 12 12 }, finish[1] = *true*

Therefore, the system is in a safe state, and a safe sequence is P0, P2, P3, P4, P1.

Thus the new request can be granted.

<u>Question 3</u>: (20 marks)

Consider a demand-paging system with the following time-measured utilizations:

CPU utilization: 20%
Paging disk utilization: 97.7%
Other I/O devices utilization: 5%

Which (if any) of the following will (probably) improve CPU utilization? Explain your answer.

<u>Solution</u>: The system obviously is spending most of its time paging, indicating over-allocation of memory. If the level of multiprogramming is reduced resident processes would page fault less frequently and the CPU utilization would improve. Another way to improve performance would be to get more physical memory or a faster paging disk.

a) Install a faster CPU.

No , because this will not reduce page fault rate.

b) Install a bigger paging disk.

No , because this will not reduce page fault rate.

c) Decrease the degree of multiprogramming.

Yes , because this will reduce page fault rate by allowing more pages of the same process to be resident in memory.

d) Install more main memory.

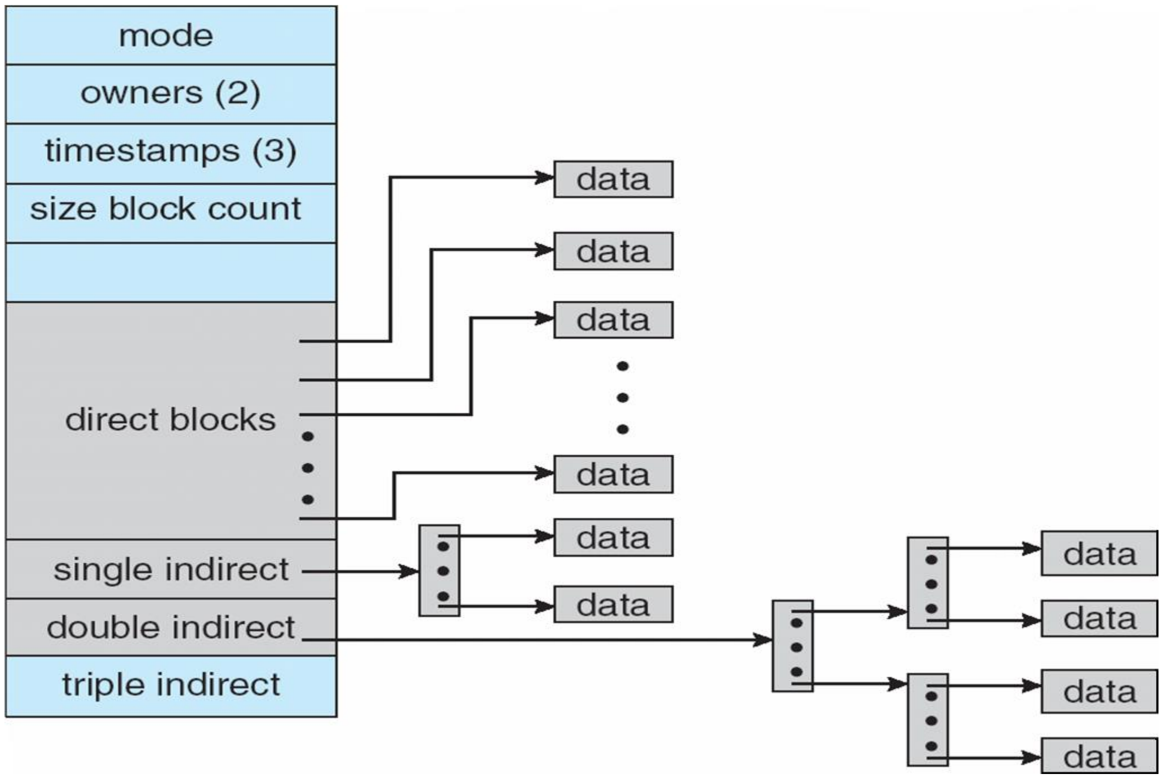
Yes , because this will reduce page fault rate by allowing more pages of the same process to be resident in memory.

e) Increase the page size.

Increasing the page size will result in fewer page faults if data is being accessed sequentially. If data access is more or less random, more paging action could result because fewer pages can be kept in memory and more data is transferred per page fault. <u>So this change is as likely to decrease utilization as it is to increase it.</u>

<u>Question 4</u>: (30 marks)

Consider the UNIX file system with the inode shown:



Knowing that the number of direct block pointers is 12, the block size is 4K bytes, and the block pointer size is 32 bits, answer the following:

a) Assuming that we won't use the triple indirect block pointer, what would be the maximum allowable file size? **[5 marks]**

<u>Solution:</u>
The 12 direct blocks can house 12 * 4 KB = 48 KB
The single indirect block has 4 KB / 4 = 1024 pointers, each pointing to a 4 KB block, amounting to 1024 * 4 KB = 4096 KB
The amount pointed at by the double indirect blocks is 1024 * 1024 * 4 KB = 4194304 KB

The total file size is 48 + 4096 + 4194304 = 4198448 KB

b) If the O.S. wants to allocate space for a 70K file, and the list of free blocks is:

5, 6, 3, 4, 8, 11, 12, 15, 16, 7, 9, 13, 14, 17, 18, 19, 20, 24, 25, 22, 23, 26, 27…

Show the contents of the block pointers in the inode. **[10 marks]**

Solution:

We need 18 blocks to house the data (18 * 4 KB = 72 KB),

Pointer #	Type	Contents
1	direct	5
2	direct	6
3	direct	3
4	direct	4
5	direct	8
6	direct	11
7	direct	12
8	direct	15
9	direct	16
10	direct	7
11	direct	9
12	direct	13
13	single indirect	14
14	double indirect	-1
15	triple indirect	-1

Block #14 has pointers to six other blocks as follows:

Pointer #	Contents
1	17
2	18
3	19
4	20
5	24
6	25
7	-1
8	-1
9	-1
	. . . .
1024	-1

c) What is the amount of internal fragmentation in b? **[5 marks]**

Solution:

There are two blocks with internal fragmentation:
The single indirect block has 6 pointers only, 6 * 4 = 24 Bytes, which leaves 4096 – 24 = 4072 Bytes of internal fragmentation.
The last data block has 2 KB of internal fragmentation.

The total internal fragmentation is 4072 + 2048 = 6120 Bytes.

d) If the file system is a DOS File Allocation Table, show the contents of the FAT for the same 70K file, and the same list of free blocks:

5, 6, 3, 4, 8, 11, 12, 15, 16, 7, 9, 13, 14, 17, 18, 19, 20, 24, 25, 22, 23, 26, 27… **[10 marks]**

Solution:

The FAT looks like this:

Entry	Contents	Entry	Contents
1		16	7
2		17	18
3	4	18	19
4	8	19	20
5	6	20	24
6	3	21	
7	9	22	
8	11	23	
9	13	24	EOF
10		25	
11	12	26	
12	15	27	
13	14	28	
14	17	29	
15	16	30	