Operating System Exercises 2010-2013

Rahmat M. Samik-Ibrahim et. al.

`http://rms46.vLSM.org/2/184.pdf`

Contributors: Rahmat M. Samik-Ibrahim (VauLSMorg), Muhammad H. Hilman, Heri Kurniawan, Amril Syalim (Faculty of Computer Science, University of Indonesia), et. al.

Table of Contents

(2011-2) Process State

There exists four (4) identical processes, with this following CPU utilization table:

	Multiprogramming Combination (%)			
	A	A+A	A+A+A	A+A+A+A
CPU utilization per proses A	10	9,5	9	8,6

The CPU time of each processes is 86 seconds

a) How long will be the total time to run concurrently all (4) processes together?!

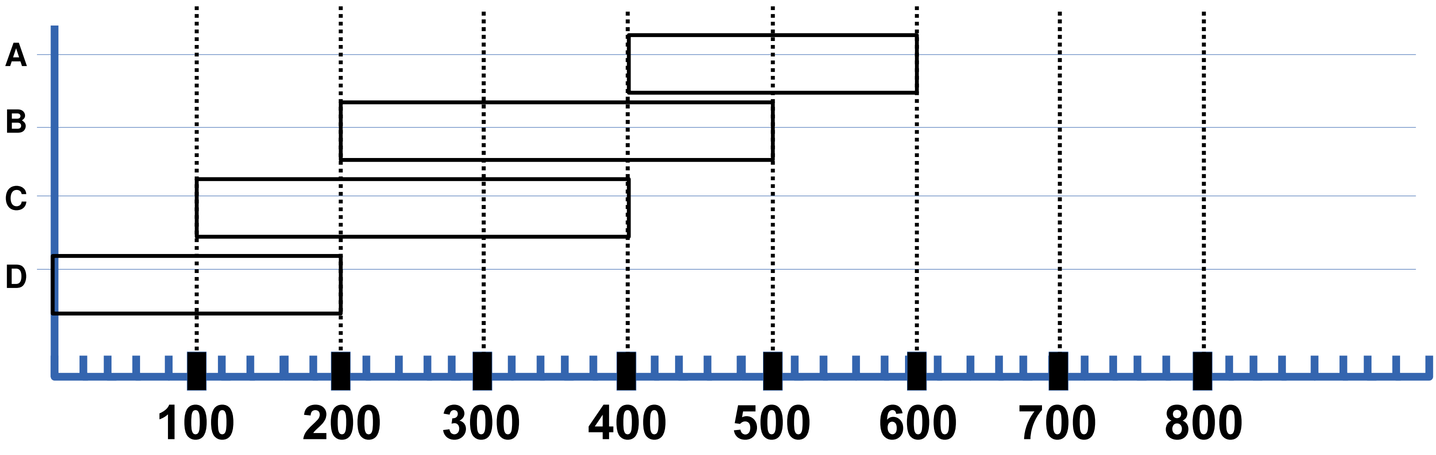
b) How long will be the total time to run all (4) processes one by one?!

(2013-2) Process State

Four (4) processes, A(90%), B(80%), C(70%), D(60%); **where [**W(X); W=process name; X= I/O Wait(%)**]** with this following CPU utilization table:

	Multiprogramming Combination (%)														
	A	B	C	D	A+B	A+C	A+D	B+C	B+D	C+D	A+B+C	A+B+D	A+C+D	B+C+D	A+B+C+D
CPU utilization per proses A	10	-	-	-	9.3	9.3	9.2	-	-	-	8.3	8.1	7.8	-	7
CPU utilization per proses B	-	20	-	-	19	-	-	18	17	-	17	16	-	15	14
CPU utilization per proses C	-	-	30	-	-	28	-	26	-	25	25	-	23	22	21
CPU utilization per proses D	-	-	-	40	-	-	37	-	35	33	-	32	31	30	28

The relation between the process and time is as following:



a) Calculate the ***CPU time*** of process A.

b) Calculate the ***CPU time*** of process B.

c) Calculate the ***CPU time*** of process C.

d) Calculate the ***CPU time*** of process D

e) What is the total execution time of process D, if it is executed alone?

(2011-2) Fork

```
#include <stdio.h>
#include <stdlib.h>
#include <semaphore.h>
#include "myutils.h"

int main(int argc, char * argv[]) {
   int ii;
   for (ii=0;ii<2;ii++) {
      fork();
      waitpid(-1,NULL,0);
   }
   printf("I am %d\n",(int) getpid());
}
```

Complete the output of that program:

```
I am 7000
```

(2013-2) Fork

```
#include <sys/types.h>
#include <sys/wait.h>
#include <stdio.h>
#include <unistd.h>

int delay1_fork (void) {
   sleep(1);               /* delay 1000 ms */
   return (int) fork();
}

void main(void) {
   int i1, i2, i3, i4, i5;

   i1 = i2 = i3 = i4 = i5 = (int) getpid();
   if (delay1_fork() > 0) {
      i1 = (int) getpid();
      if (delay1_fork() == 0) {
         i2 = (int) getpid();
         if (delay1_fork() > 0) {
            i3 = (int) getpid();
            if (delay1_fork() == 0) {
               i4 = (int) getpid();
               if (delay1_fork() > 0) {
                  i5 = (int) getpid();
                  sleep (1);
               }
            }
         }
      }
   }
   printf ("i1=%d - i2=%d - i3=%d - i4=%d - i5=%d \n", i1, i2, i3, i4, i5);
   fflush(stdout);
   wait(NULL);
   wait(NULL);
   wait(NULL);
}
```

a) Convert your right most student ID into column [A-E]: **[0 → A], [1 → B], [2-3 → C], [4-5 → D], [>5 → E]**!

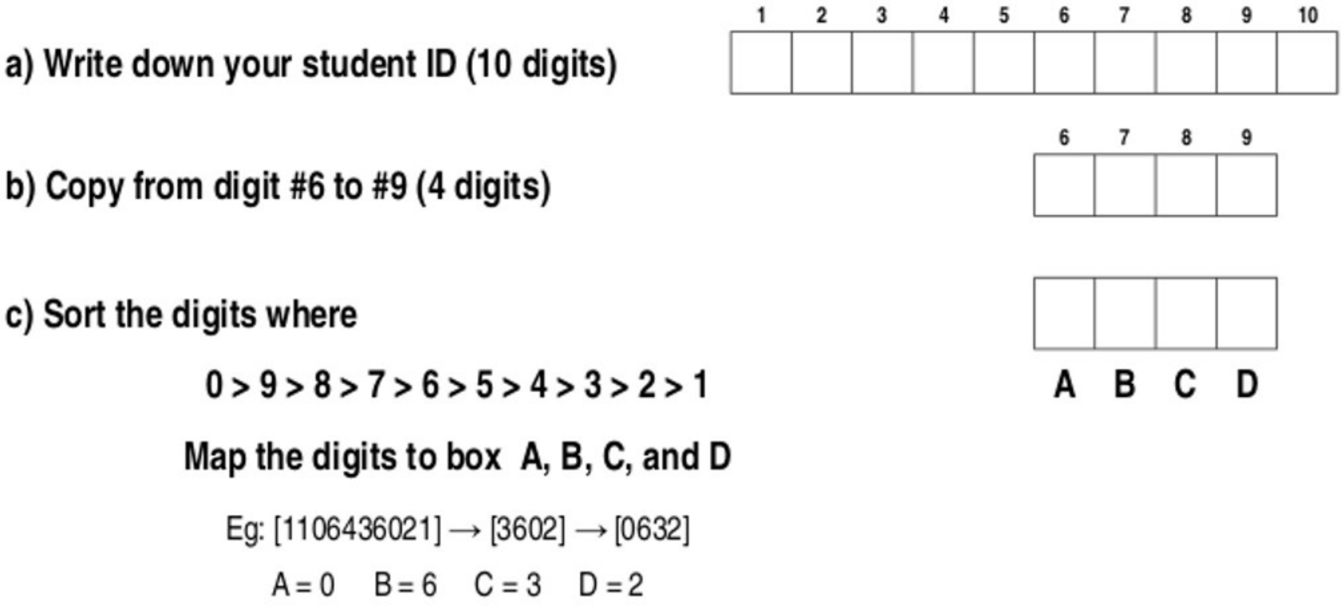
b) Convert your entrance year into row[I-VI]:**[<2009→I], [2009→II], [2010→III], [2011→IV], [2012→V], [2013→VI]**!

c) Fill the value “1000” into the column/row below based on (a) and (b).

d) Fill the rest below based on **the output of the program above**!

	A [0]	**B** [1]	**C** [2][3]	**D** [4][5]	**E** [> 5]
I (<2009)	i1=	i2=	i3=	i4=	i5=
II (2009)	i1=	i2=	i3=	i4=	i5=
III (2010)	i1=	i2=	i3=	i4=	i5=
IV (2011)	i1=	i2=	i3=	i4=	i5=
V (2012)	i1=	i2=	i3=	i4=	i5=
VI (2013)	i1=	i2=	i3=	i4=	i5=

(2012-2) Thread Scheduling



a) Write down your student ID (10 digits)

1	2	3	4	5	6	7	8	9	10

b) Copy from digit #6 to #9 (4 digits)

6	7	8	9

c) Sort the digits where

0 > 9 > 8 > 7 > 6 > 5 > 4 > 3 > 2 > 1

Map the digits to box A, B, C, and D

A	B	C	D

Eg: [1106436021] → [3602] → [0632]

A = 0   B = 6   C = 3   D = 2

The mapping of function “**`day_month(X)`**” is as following:

X	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**0**
day_month(X)	**0**	**31**	**59**	**90**	**120**	**151**	**181**	**212**	**243**	**273**

At the end of executing function “**`init_n_start_thread(T1,T2,....Tn)`**”, will appear “**n**” ***threads non-preemptive*** T1, T2, … Tn. Each ***thread*** will be put into the “***Ready Queue***” as “**Rd**” (Ready). If the CPU is available, the state will be changed to “**R**” (Run). There will be no new thread be executed before the threads from the previous **`init_n_start_thread()`** are done.

This following is the unit time table for the “***Pseudo-program***” of the following page.

No	***Type***	***Unit***	***Example (line)***
1	M (Main function)	1 (R)	**`01 M: {`**
2	T (Thread)	1 (R)	**`05 T1:{`**
3	One Line Executions	1 (R) / line	**`16        day1 = dm1 + D;`**

4	Function Executions	2 (RR) / line	**`10        dm1 = day_month(B);`**
			`20        init_n_start_thread(T9);`

d) Calculate:

delta = ________

```
##  This program consists of M: (main) and T1...T9 (threads).
##  Fill A, B, C, D according your student ID. See the previous page.

01 M: {
02       A = ________ ;
03       B = ________ ;
04       init_n_start_thread(T1, T2, T3, T4, T5);
      }
##
05 T1:{
06       C = ________ ;
      }
##
07 T2:{
08       D = ________ ;
      }
##
09 T3:{
10       dm1 = day_month(B);
      }

##
11 T4:{
12       dm2 = day_month(A);
      }

##
13 T5:{
14       init_n_start_thread(T6, T7, T8);
      }

##
15 T6:{
16       day1 = dm1 + D;
      }

##
17 T7:{
18       day2 = dm2 + C;
      }

##
19 T8:{
20       init_n_start_thread(T9);
      }

##
21 T9:{
22       delta = day2 - day1;
      }
```

Fill / complete this diagram for ONE PROCESSOR (P1). See the examples.

T	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
Line-P1	01	02	03	04	04	05	06	07	08	09																				
P1	M	M	M	M	M	T1	T1	T2	T2	T3																				
M	R	R	R	R	R	-	-	-	-	-																				
T1	-	-	-	-	-	R	R	-	-	-																				
T2	-	-	-	-	-	Rd	Rd	R	R	-																				
T3	-	-	-	-	-	Rd	Rd	Rd	Rd	R																				
T4	-	-	-	-	-	Rd	Rd	Rd	Rd	Rd																				
T5	-	-	-	-	-	Rd	Rd	Rd	Rd	Rd																				
T6	-	-	-	-	-	-	-	-	-	-																				
T7	-	-	-	-	-	-	-	-	-	-																				
T8	-	-	-	-	-	-	-	-	-	-																				
T9	-	-	-	-	-	-	-	-	-	-																				

Fill / complete this diagram for TWO PROCESSORS (P1+P2). See the examples.

T	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
Line-1	01	02	03	04	04	05	06	09	10	10																				
P1	M	M	M	M	M	T1	T1	T3	T3	T3																				
Line-2	-	-	-	-	-	07	08	11	12	12																				
P2	-	-	-	-	-	T2	T2	T4	T4	T4																				
M	R	R	R	R	R	-	-	-	-	-																				
T1	-	-	-	-	-	R	R	-	-	-																				
T2	-	-	-	-	-	R	R	-	-	-																				
T3	-	-	-	-	-	Rd	Rd	R	R	R																				
T4	-	-	-	-	-	Rd	Rd	R	R	R																				
T5	-	-	-	-	-	Rd	Rd	Rd	Rd	Rd																				
T6	-	-	-	-	-	-	-	-	-	-																				
T7	-	-	-	-	-	-	-	-	-	-																				
T8	-	-	-	-	-	-	-	-	-	-																				
T9	-	-	-	-	-	-	-	-	-	-																				

Fill / complete this diagram for FIVE PROCESSORS (P1, P2,... P5). No example.

T	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
Line-1																														
P1																														
Line-2																														
P2																														
Line-3																														
P3																														
Line-4																														
P4																														
Line-5																														
P5																														
M																														
T1																														
T2																														
T3																														
T4																														
T5																														
T6																														
T7																														
T8																														
T9																														

(2011-2) Synchronization

Analyze this following **"myutils.h"** header program:

```
#define MAX_THREAD 100
#define BUFFER_SIZE  5
#define TRUE         1
#define FALSE        0
typedef struct {
   int   buffer[BUFFER_SIZE];
   int   in;
   int   out;
   int   count;
} bbuf_t;
void daftar_trit   (void* trit);      // mempersiapkan "trit"
void jalankan_trit (void);            // menjalankan dan menunggu hasil dari
                                      // "daftar_trit"
void beberes_trit  (char* pesan);     // beberes menutup "jalankan_trit"
void rehat_acak    (long max_mdetik); // istirohat acak "0-max_mdetik" (ms)
void init_buffer   (void);            // init buffer
void enter_buffer  (int entry);       // enter  an integer item
int remove_buffer  (void);            // remove the item
void init_rw       (void);            // init readers writers
int  startRead     (void);            // start reading
int  endRead       (void);            // end reading
void startWrite    (void);            // start writing
void endWrite      (void);            // end writing
```

```
#include <stdio.h>
#include <stdlib.h>
#include <semaphore.h>
#include "myutils.h"
sem_t                  mutex1, mutex2;

void* THREAD_one (void* a) {
   sem_wait (&mutex1);
   printf("This is THREAD one\n");


}

void* THREAD_two (void* a) {
   sem_wait (&mutex2);
   printf("This is THREAD two\n");
   sem_post (&mutex1);
}

void* THREAD_three (void* a) {

   printf("This is THREAD three\n");

   sem_post (&mutex2);

}

int main(int argc, char * argv[]) {
   sem_init (&mutex1, 0, 0);
   sem_init (&mutex2, 0, 0);


   daftar_trit(THREAD_one);
   daftar_trit(THREAD_two);
   daftar_trit(THREAD_three);

   jalankan_trit();
   beberes_trit("Good Bye");
}
```

a) Write down the output of the program:

____

____

____

____

____

____

b) Complete/modify the program by adding **"THREAD_four" (thread)** which will print "This is THREAD four" after THREAD_one.

(2012-2) Synchronization

Add the pseudo-program in "**(2012-2) Thread Scheduling**" with synchronization semaphores, so that there will be no new thread be executed before the threads from the previous **`init_n_start_thread()`** are done.

Add to the pseudo-program, the combination of these following three semaphore functions:

`sem_init(X,Y)    – initial semaphore` "X" with value "Y"

`sem_wait(X)      – semaphore` "X" wait

`sem_signal(X)    – semaphore` "X" signal

(2013-2) Synchronization

The program (next page) is using function like “**`daftar_trit()`**” for registering a “*threat*” and “**`jalankan_trit()`**” for starting the “*threats*”. The “**`sem_init()`**” function will initiate a *semaphore*, “**`sem_post()`**” to signal a *semaphore*, and “**`sem_wait()`**” to wait for a *semaphore*.

a) Write down one posibility of the program output.

b) From the output above, which lines might be in different sequences. Explain!

c) What is the role of semaphore “**mutex**”?

d) What is the role of semaphore “**switch1**”?

e) What is the role of semaphore “**switch2**”?

```
/*
 * $Revision: 140 $
 * (c) 2013 Rahmat M. Samik-Ibrahim
 * This is FREE SOFTWARE.
 */

#include <stdio.h>
#include <stdlib.h>
#include <semaphore.h>
#include "myutils.h"

#define      NThreads 4

sem_t        mutex,   switch1, switch2;
int          addvar1, addvar2, addresult;
int          subvar1, subvar2, subresult;
int          mulvar1, mulvar2, mulresult;
int          divvar1, divvar2, divresult;

void* manager (void* a) {
   printf("Manager starts \n");

   for (int ii=0; ii< NThreads;ii++)
       sem_wait (&switch1);
   sem_wait  (&mutex);
   addvar1 = 5;
   addvar2 = 2;
   subvar1 = 7;
   subvar2 = 2;
   mulvar1 = 2;
   mulvar2 = 3;
   divvar1 = 4;
   divvar2 = 2;
   sem_post  (&mutex);

   for (int ii=0; ii< NThreads;ii++)
       sem_post (&switch2);
   for (int ii=0; ii< NThreads;ii++)
       sem_wait (&switch1);
   printf("Result add:%d; sub:%d; mul:%d;
      div:%d;\n",
      addresult, subresult, mulresult,
      divresult);
}

void* add (void* a) {
   sem_post  (&switch1);
   sem_wait  (&switch2);

   sem_wait  (&mutex);
   printf("Add starts \n");
   addresult = addvar1 + addvar2;
   sem_post  (&mutex);
   sem_post  (&switch1);
}

void* subtract (void* a) {
   sem_post  (&switch1);
   sem_wait  (&switch2);

   sem_wait  (&mutex);
   printf("Subtract starts \n");
   subresult = subvar1 - subvar2;
   sem_post  (&mutex);
   sem_post  (&switch1);
}

void* multiply (void* a) {
   sem_post  (&switch1);
   sem_wait  (&switch2);
   sem_wait  (&mutex);
   printf("Multiply starts \n");
   mulresult = mulvar1 * mulvar2;
   sem_post  (&mutex);
   sem_post  (&switch1);
}

void* divide (void* a) {
   printf("Divide starts \n");
   sem_post  (&switch1);
   sem_wait  (&switch2);
   sem_wait  (&mutex);
   divresult = divvar1 / divvar2;
   sem_post  (&mutex);
   sem_post  (&switch1);
}

void main(void) {
   sem_init      (&mutex,   0, 1);
   sem_init      (&switch1, 0, 0);
   sem_init      (&switch2, 0, 0);
   daftar_trit   (manager);
   daftar_trit   (add);
   daftar_trit   (subtract);
   daftar_trit   (multiply);
   daftar_trit   (divide);
   jalankan_trit ();
   beberes_trit  ("Done...");
}
```

(2012-2) Deadlock

a) What are the four conditions that can arise a ***deadlock*** situation. Explain each condition in 1-2 sentences.
b) What are the three ways that are generally used to handle ***deadlock***? Explain each way in 1-2 sentences.
c) Which solution is the most used way to handle ***deadlock***? Explain it in 1-2 sentences.

(2011-2/UCB sp00mt2) Demand Paging

Consider a demand paging system with four physical memory frames and the following reference string over seven pages:

1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 2, 3, 7, 6

Assuming that memory starts empty, **how many page faults** will occur and what **pages** will be the **final contents of memory** under:

a) the FIFO page replacement policy?
b) the LRU page replacement policy?
c) the OPTIMAL page replacement policy?

(2011-2/UCB fa08mt1) Memory

Suppose we have a 32-bit virtual addresses system with 2 TB physical addresses. The page size is 8 kB. The Page Table Entry (PTE) will consist of:

- some **bits** for all frames of the physical address;
- one Valid/Invalid **bit;** and
- some unused **bit**s.

a) What will be the frame size (1 TB = 1024 GB = 1024 x 1024 MB)?
b) In how many frames will the physical address be divided?
c) How many bits are needed in the PTE for the frame part?
d) Draw a complete PTE scheme with the “frame numbers” bits, “valid/invalid” bit, and “unused” bits. How many bytes (rounded) are needed?

e) How many PTEs can be fit into one virtual page?
f) Draw a complete virtual address scheme with “page numbers” bits and “offset” bits.

g) How many PTEs are needed to map the whole the virtual address space?
h) How many virtual pages are needed for those all PTEs above?

(2013-2) Memory

Consider a system with a 16 bits Virtual Address space and a 24 bits Physical Address space. The offset/ displacement ialah 12 bits. The Page Table starts at Physical Address 001000 (HEX).

Address (HEX)	***+0***	***+1***	***+2***	***+3***	***+4***	***+5***	***+6***	***+7***	***+8***	***+9***	***+A***	***+B***	***+C***	***+D***	***+E***	***+F***
001000	10	24	10	23	10	22	10	21	10	20	10	1F	10	1E	10	1D
001010	10	34	10	33	10	32	10	31	10	30	10	2F	10	2E	10	2D
……																
01F000	A0	A1	A2	A3	A4	A5	A6	A7	A8	A9	AA	AB	AC	AD	AE	AF
……																
020000	B0	B1	B2	B3	B4	B5	B6	B7	B8	B9	BA	BB	BC	BD	BE	BF
……																
023000	C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	CA	CB	CC	CD	CE	CF
……																
031000	D0	D1	D2	D3	D4	D5	D6	D7	D8	D9	DA	DB	DC	DD	DE	DF
……																

The **TLB** entry is (in HEX): "**`[Page Number][Flag 4 bits][Frame Number]`**". The "Frame Number" is INVALID unless the FLAGS value is "1 (HEX)" or "0001 (BIN)".

`[Page#]`	`[Flags]`	`[Frame#]`		
0	1	0	2	4
1	1	0	2	3
2	1	0	2	2
3	1	0	2	1
4	1	0	2	0

a) Write down your student ID (NPM):

b) Write down the most right digit of your student ID: _________

c) VA1 is 1000 (HEX) plus the value of point "b":   VA1 = ____ + 1000 (HEX) = ______

d) VA2 is 5000 (HEX) plus the value of point "b":   VA2 = ____ + 5000 (HEX) = ______

e) Draw the 16 bit Virtual Address diagram:

Page Number (?? bit)	Offset (12 bit)

How many bits are there in the *"Page Number"* space? ____________ bit

f) Based on the Page Number space, how many pages are there? _________ pages

g) Draw the 24 bit Physical Address diagram:

Frame Number (?? bit)	Offset (12 bit)

How many bits are there in the *"Frame Number"* space? ____________ bit

h) Based on the Frame Number space, how many frames are there? _________ frames

i) What is the Page Number of VA1 (see point "c"): __________________________

j) Is VA1's *Frame Number* in TLB ? Yes / No (circle the correct one)

k) Is VA1's *Frame Number* in PT (*Page Table*) ? Yes / No

l) The *Frame Number* for VA1 is Valid / Not Valid / Not Found

m) If valid, what is the *Frame Number* value for VA1? __________________________

n) (If applicable) What is the Physical Address (PA1) value of VA1? __________________________

o) (If applicable) What number is inside VA1 __________________________

p) What is the Page Number of VA2 (see point "d"): __________________________

q) Is VA2's *Frame Number* in TLB ? Yes / No (circle the correct one)

r) Is VA2's *Frame Number* in PT (*Page Table*) ? Yes / No

s) The *Frame Number* for VA2 is Valid / Not Valid / Not Found

t) If valid, what is the *Frame Number* value for VA2? __________________________

u) (If applicable) What is the Physical Address (PA2) value of VA2? __________________________

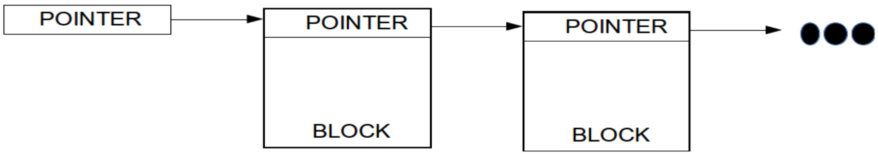
v) (If applicable) What number is inside VA2 __________________________

(2011-2/Wikipedia) Disk IBM 350 Storage Unit

The IBM 350 disk storage unit model 1 (151 cm x 171 cm x 50.2 cm) was announced on September 4, 1956. IBM 350 had 100 surfaces @ 100 tracks. Each track had 5 sectors @ 600 bits. The disk spun at 1200 RPM. The seek time was about (10 + 10 Tr) mS where “Tr” was the Track differences. Switching time between surfaces was assumed 0 mS. At that time, the main purpose of IBM 350 was to store “6 bits” characters.

a) How many "6 bits” characters can be stored into one sector?
b) How many "6 bits” characters can be stored into one track?
c) How many "6 bits” characters can be stored into one cylinder?
d) How many "6 bits” characters can be stored into the whole disk?
e) Calculate the maximum theoretical data transfer rate in “characters per seconds”.
f) How long (mS) will it take to write in sequence (sector by sector), 50100 characters starting from (Track 0, Surface 0, Sector 0) or (0, 0, 0) to (0, 0, 4) to (0, 1, 0) to (0, 1, 4) to (0, 2, 0) and so on.
g) Which one (Track, Surface, Sector) will be the last to be written?

(2013-2) Disk



Consider a disk with 256 M sectors. Design a File System with linked-list blocks. There will be a pointer to the next block inside each block. The pointer size should be bytes (1, 2, 3, or more). The size of one block is equal to the size of a sector is equal to 8 kBytes. Use 1k=1024; 1M=1024k; 1G=1024M; 1T=1024G.

a) What is the size of the disk (in Tbytes)?
b) What is the size of the pointer (in bits)?
c) Explain, why the maximum size of a file will be always less than the size of the disk!

(2013-2) RAID

Consider a collections of 1 Tbytes disks. Design a disk array with performance at least six times better compared to an individual disk. The amount of disks should be as few as possible.

a) Draw a diagram of a disk array without any fault tolerance. How many disks will be used?
b) Draw a diagram of a RAID 6 + 0. How many disks will be used?

(2012-2) Performance

Consider an application with total execution time of 100 hours. Based on an analysis, 24% of execution time is the CPU time and the rest is mostly the time of handling the HDD. Suppose you would like to make the application execution time less than 50 hours. First, the CPU is replaced with a one that has got 2 (two) cores where each core is twice faster. Second, a new RAID 6+0 configuration will be deployed, however using the same HDD type as before.

a) Draw a diagram of the RAID 6+0 disks that fulfill the requirements!
b) How much will be the minimum disk capacity of the new HDD configuration?