Operating System Exercises 2008-2010

Rahmat M. Samik-Ibrahim and Heri Kurniawan

http://rms46.vLSM.org/2/171.pdf

Contributors: Rahmat M. Samik-Ibrahim (VauLSMorg) and Heri Kurniawan (Faculty of Computer Science, University of Indonesia).

Table of Contents

Short Questions

a) Explain briefly, the **two basic functions** that Operating Systems perform!
b) One of the Operating Systems' basic function is to present the user with the equivalent of an extended machine. Explain what an **extended machine** is!
c) What is a **virtual machine**? Give an example/illustration!
d) One of the Operating Systems' basic function is managing resources. Explain what an **managing resources** is!
e) These following are fundamental principles of an Operating System:
(a) Processes, (b) Interprocess Communications, (c) Semaphores, (d) Message Passing, (e) Schedulling Algorithm, (f) Input/Output (g) Deadlocks, (h) Device Drivers, (i) Memory Management, (j) Paging Algorithm, (k) File Systems, and (l) Security & Protections.
f) Explain briefly three (3) fundamental principles from the list above!
g) What is a **Real Time** system? Give an illustration!
h) What is a **Hard Real Time** system? Give an illustration!
i) What is a **Soft Real Time** system? Give an illustration!
j) What are the differences between a **System Program** and an **Application Program**?
k) Give an example of a **System Program**!
l) Give an example of a **Application Program**!
m) What are the differences between a **System Program** and a **System Call** ? Give an illustration (eg. "creating a directory")!
n) How is Win32 API (Application Program Interface) related to a **System Call** .
o) Explain what a **Critical Region** is!
p) Explain what a **Race Condition** is!
q) Explain what a **Busy Waiting** is! How to overcome it?
r) What is a **Deadlock** ? Explain briefly!
s) How does Unix handle the **Deadlock** problem? Explain briefly!
t) What is a **Starvation** ? Explain briefly!
u) How does these following systems handle the **deadlock** problem:
i) Unix
ii) Windows
iii) JVM
Explain briefly!
v) What is a binary semaphore?
w) Explain briefly, how to use binary semaphores for access control of a critical section!
x) What is a counting semaphore?
y) Explain briefly, how to use counting semaphores for access control of a resource with a finite number of instances?
z) Explain the differences between running a process in **kernel mode** and **user mode**?
aa) Give two examples/illustration of running a process in "kernel mode".
ab) Give two examples/illustration of running a process in "user mode".
ac) Explain what "multi-programming" means. Give an example!
ad) Explain what "multi-users" means. Give an example!
ae) Explain what a "process table" is. Give an illustration!
af) Explain what a "file system" is. Give an example!
ag) Explain what a "pipe" is. Give an illustration!
ah) Explain what a "socket" is. Give an illustration!
ai) In a three state process model ("running", "blocked", and "ready"), explain briefly about each

process state.

aj) In a three state process model ("running", "blocked", and "ready”), explain why a "running" state process transits to "blocked" state.
ak) In a three state process model ("running", "blocked", and "ready”), explain why a "running" state process transits to "ready" state.
al) In a three state process model ("running", "blocked", and "ready”), explain why a "ready" state process transits to "running" state.
am) In a three state process model ("running", "blocked", and "ready”), explain why a "blocked" state process transits to "ready" state.
an) In a three state process model ("running", "blocked", and "ready”), explain why there is no "blocked" state process transits to "running" state.
ao) In a three state process model ("running", "blocked", and "ready”), explain why there is no "ready" state process transits to "blocked" state.
ap) What is a "CPU bound" process? Give an illustration!
aq) What is a "I/O bound" process? Give an illustration!
ar) What is a "preemptive" process? Give an illustration!
as) What is a "non-preemptive" process? Give an illustration!
at) Explain briefly the "Readers/Writers" problem. How to avoid "deadlock" in the "Readers/Writers" problem.
au) Explain briefly the "Readers/Writers" problem. Where is the "critical section" of the "Readers/Writers" problem.
av) Explain briefly the "Consumer/Producer" problem. How to avoid "deadlock" in the "Consumer/Producer" problem.
aw) Explain briefly the **Consumer**/**Producer** problem. Where is the **critical section** of the **Consumer**/**Producer** problem.
ax) What are the differences and similarities between the **Consumer**/**Producer** problem and **Readers**/**Writers** problem? Explain briefly!
ay) Explain how a "preemptive" system can improve performance!
az) What will improve, if more "RAM" is added to a system? Give illustrations!
ba) What will improve, if the "CPU" of the system is replaced with a faster one? Give illustrations!
bb) What will improve, if the "DISK" of the system is replaced with a faster transfer rate? Give illustrations!
bc) What will improve, if the "I/O Bus" of the system is replaced with a faster transfer rate? Give illustrations!
bd) Which task should have more priority: writing to a disk or reading from a disk? Explain!
be) Explain how a higher "RPM rate" can improve disk transfer rate!
bf) Explain how a higher "disk density" can improve disk transfer rate!
bg) Explain how a DMA scheme can improve the system performance!
bh) What is a "Hard Real Time System"? Give an example!
bi) What is a "Soft Real Time System"? Give an example!
bj) Compare the performance between a "pipe" and "file". Explain!
bk) Compare the performance between a "pipe" and "socket". Explain!
bl) Compare the performance between a "socket" and "file"? Explain!

Operating Systems Concepts

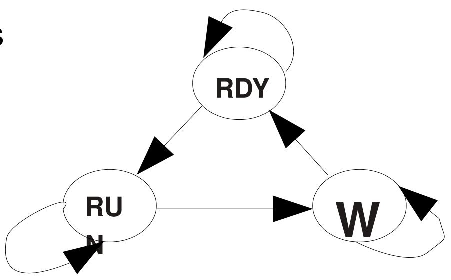
a) Describe briefly, what a system program is!
b) Give some examples of point 'a'.
c) Describe briefly, what a system call is!
d) Give some examples of point 'c'.
e) Is “disk format” a system program or a system call? Explain!

Intellectual Property Rights

a) Describe briefly the similarities between the Open Source Software and the Free Software concept!
b) Describe briefly the differences between the Open Source Software and the Free Software concept!
c) Describe briefly the differences between Free Software and Copyleft!
d) Give an example of a Free Software that is not Copyleft!

Process State I

- At t=0, all processes (P $_1$, P $_2$, P $_3$, P $_4$) are in the ' **' RDY** " state.
- The "**RUN**/**W**" (Wait) state patterns of each process are as following:
  - P_1 (2, 9, 2, 9, 2, 9, ...)
  - P_2 (1, 9, 1, 9, 1, 9, ...)
  - P_3 (2, 6, 2, 6, 2, 6, ...)
  - P_4 (1, 6, 1, 6, 1, 6, ...)



- Only one process can be in the "**RUN**" state at any time.
- Many processes can be in the "**W**" and/or "**RDY**" states.
- The "**RDY**" to "**RUN**" transition rules are as following:
  - Priority is for the process with the **shortest** waiting time (from recent arrival in "**RDY**").
  - If "tie", priority is given to the process with the **smallest** index.
  - If "**RUN**" is empty, a process can directly transit from "**W**" via "**RDY**" to "**RUN**".

a) Please fill the first 25 time units of this following Gantt Chart:
   i) The state of each processes (P_1, P_2, P_3, P_4).
   ii) Which process is in the RUN state (RUN).
   iii) How many processes are in the Ready state (RDY).

RDY	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
P1																									
RDY	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
P2																									
RDY	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
P3																									
RDY	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
P4																									
	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
RUN																									
	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
RDY																									

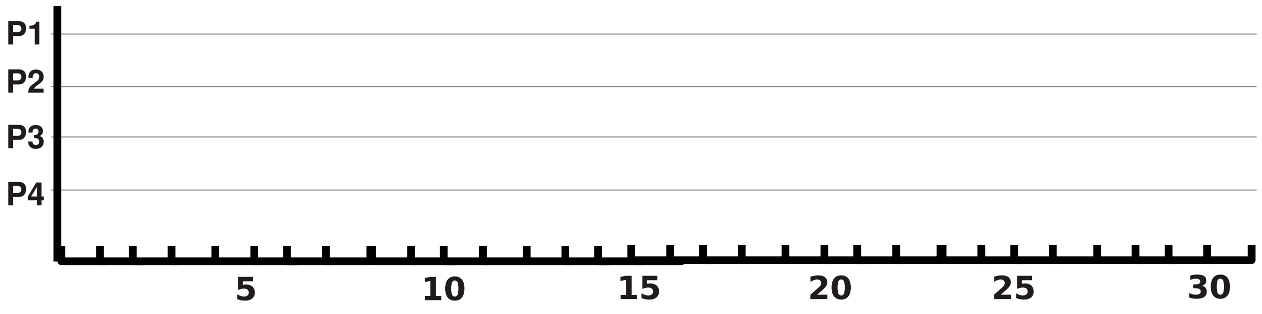
b) Calculate (in %), how much the **CPU** utilization is.
c) What is the average load (in %) of the RDY state?

Process State II

There exists four processes, P1(0:2.0), P2(5:4.9), P3(10:2.9), P4(15:3.3); [where P**n**(**A**:**B**) means **n**=process number; **A**=starting time; **B**=CPU time] with this following CPU utilization table:

(I/O Wait = 60%)	*Multiprogramming Degree*			
	1	*2*	*3*	*4*
CPU busy	40%	64%	78%	88%
CPU busy per process	40%	32%	26%	22%

Please draw a "processes/time relation" chart:



Process State III

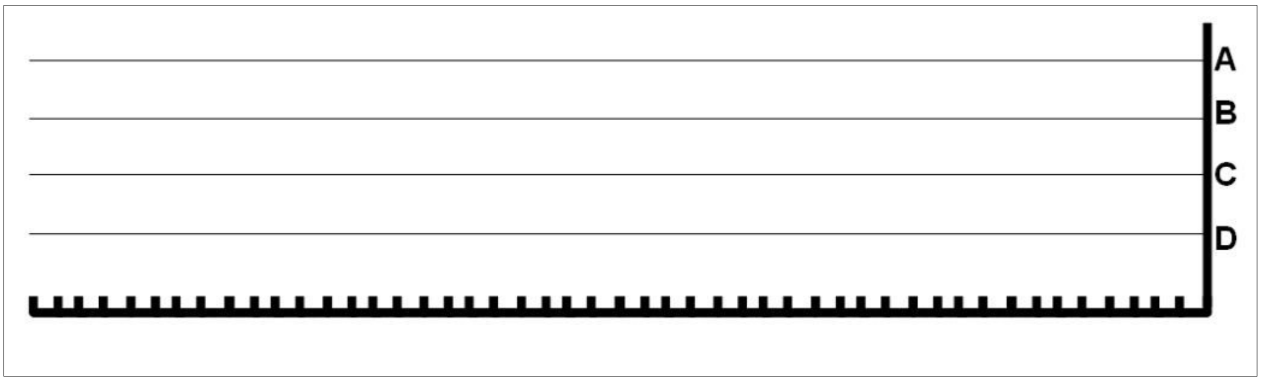
(See Process State II) There exists four processes, P1(0:4.0), P2(10:4.9), P3(15:2.9), P4(20:3.3); [where P**n**(**A**:**B**) means **n**=process number; **A**=starting time; **B**=CPU time].

Process State IV

There exists four processes, A(90: 34.6), B(80: 50), C(70: 46), D(60: 28); [where **X**(**Y**:**Z**) means **X**=process; **Y**=I/O Wait (%); **Z**=CPU time] 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 (process A)	10	-	-	-	9.3	9.3	9.2	-	-	-	8.3	8.1	7.8	-	7
CPU utilization (process B)	-	20	-	-	19	-	-	18	17	-	17	16	-	15	14
CPU utilization (process C)	-	-	30	-	-	28	-	26	-	25	25	-	23	22	21
CPU utilization (process D)	-	-	-	40	-	-	37	-	35	33	-	32	31	30	28

All processes terminate at the same time. Please draw a "processes/time relation" chart and calculate the starting time of all processes!



Process State V

(See Process State IV) There exists four processes, A(90: 0: 1), B(80: 10: 8), C(70: 20: 4.8), D(60: 30: 6.5); [where **P(X:Y:Z)** means **P** =process; **X** =I/O Wait (%); **Y** = arrival time ; **Z** =CPU time] with this following CPU utilization table above. Please draw a process/time relation at the diagram above.

Process State VI

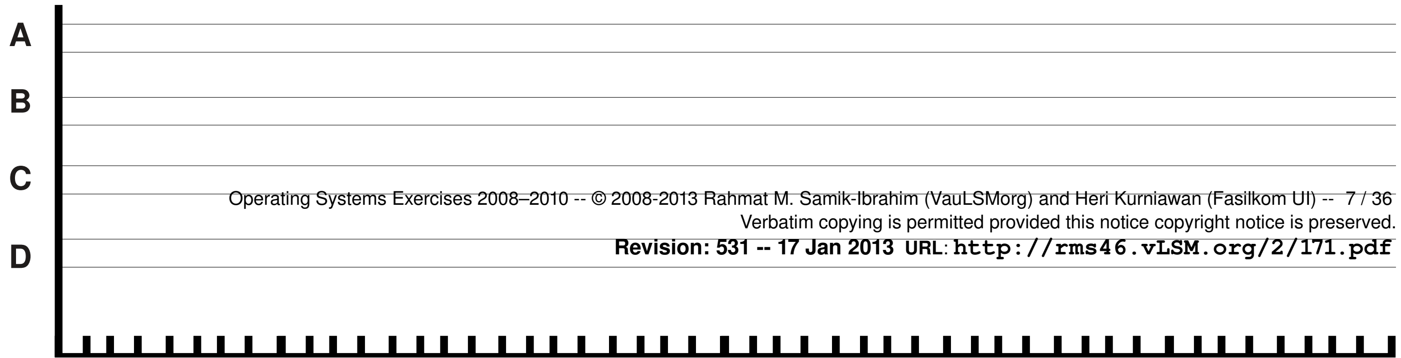
There exists four processes, A(90: 150: 7), B(80: 100: 29), C(70: 50: 68), D(60: 0: 131); [where **P(X:Y:Z)** means **P** =process; **X** =I/O Wait (%); **Y** =arrival time; **Z** =CPU time] with this following CPU utilization table above. Please draw a process/time relation at the diagram above.

Process State VII (2009)

There exists four processes, A(90: 0: 1.7), B(80: 10: 8.7), C(70: 10: 6.9), D(60: 10: 5.8); [where **P(X:Y:Z)** means **P**=process; **X**=I/O Wait (%); **Y**=arrival time; **Z**=CPU time] 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

Please draw a process/time relation at following:



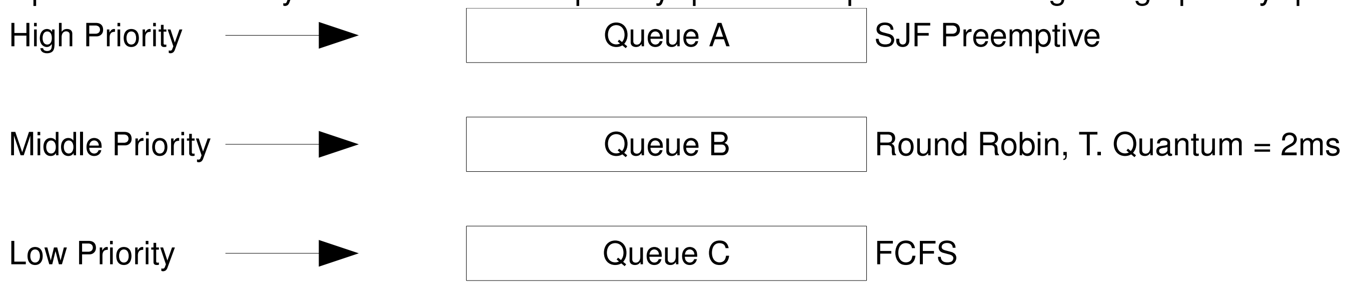
CPU Scheduling

An Operating System uses multilevel queue to schedule the processes execution. A multilevel queue scheduling consists of three queues ordered by priority level (high, middle, and low). Each queue has different scheduling algorithm.

- First Queue (Queue A) uses SJF Preemptive scheduling and eligible for process A1, A2, …, An
- Second Queue (Queue B) uses Round Robin scheduling with time quantum = 2ms and eligible for process B1, B2, …, Bn
- Third Queue (Queue C) uses First Come First Serve (FCFS) scheduling and eligible for process C1, C2, …, Cn

CPU will execute those queues under the following rules:

- CPU execute the queue based on its priority. If each queue is not empty, the queue that has high priority (Queue A) will be executed first. After that, CPU executes middle priority queue, then low priority queue.
- If high priority queue is empty, CPU executes process at other less priority queue
- If there is a process entering an empty high priority queue while CPU is executing process in other less priority queue, CPU must change its execution to high priority queue to service the process. CPU may move to other less priority queue if no process waiting in high priority queue.



Process	***A1***	***A2***	***A3***	***A4***	***B1***	***B2***	***B3***	***C1***	***C2***	***C3***
Arrival Time	0	2	8	11	2	4	9	3	5	7
Burst Time (ms)	4	2	1	4	5	4	3	7	8	2

Draw the gantt chart and determine the **total waiting time** of each queue. BE CAREFULL in writing the process starting time on each Gantt chart.

Queue A (QA)

Total Waiting Time (QA) =

Queue B (QB)

Total Waiting Time (QB) =

Queue C (QC)

Total Waiting Time (QC) =

Fork I

```
/************************************************************/
/* doublefork (c) 2006 Rahmat M. Samik-Ibrahim, GPL-like    */
/************************************************************/
#include <stdio.h>
#include <stdlib.h>
/**************************************************** main ***/
main()
{
   int pid1, pid2, pid3, pid4;

   pid1=(int) getpid();         /* what is my PID ?          */
   pid2=(int) fork();           /* split parent and child    */
   wait (NULL);                /* parent wait for its child */
   pid3=(int) fork();
   wait (NULL);
   pid4=(int) getpid();
   printf("[%4d] [%4d] [%4d] [%4d]\n", pid1, pid2, pid3, pid4);
}
/*************************************************************/
```

Suppose the process ID (PID) is 5000. Assume that each new child process will have the next sequential PID that is available (5001, 5002, etc.). Please write down the output of these processes!

Fork II

Please write down the output of this following C-program "isengfork1"!

```
/*******************************************************/
/* isengfork1 (c)2007 Rahmat M. Samik-Ibrahim, GPL-like */
/*******************************************************/
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>
main(){
   int ii=0;
   if (fork() >  0) ii++;
   wait(NULL);
   if (fork() == 0) ii++;
   wait(NULL);
   if (fork() <  0) ii++;
   wait(NULL);
   printf ("Result = %3.3d \n",ii);
}
/ ******************************************************/
```

Fork III

Please write down the output of this following C-program "isengfork1"!

```
/*******************************************************/
/* isengfork2 (c)2008 Rahmat M. Samik-Ibrahim, GPL-like */
/*******************************************************/
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>
main()
{
   int ii=2;
   if (fork() >  0) ii--;
   wait(NULL);
   if (fork() == 0) ii--;
   wait(NULL);
   if (fork() <  0) ii--;
   wait(NULL);
   printf ("Result = %3.3d \n",ii);
}
/*******************************************************/
```

Fork IV

```
01 /* cascafork (c) 2008 Rahmat M. Samik-Ibrahim, GPL-like      */
02 /********************************************************* */
03 #include <sys/types.h>
04 #include <sys/wait.h>
05 #include <stdio.h>
06 #include <stdlib.h>
07 #include <unistd.h>
08 #define DISPLAY "This is PID[%5.5d]\n"
09 /************************************************ main ** */
10 main(void) {
11    if (fork()          != (pid_t) 0) {
12       sleep(1);
13       if (fork()       == (pid_t) 0) {
14          sleep(1);
15          if (fork()    != (pid_t) 0) {
16             sleep(1);
17             if (fork() == (pid_t) 0) {
18                sleep(1);
19             }
20          }
21       }
22    }
23    printf(DISPLAY, (int) getpid());
24    waitpid(-1,NULL,0);
25    waitpid(-1,NULL,0);
26    exit (0);
27 }
28 /*********************************************************/
```

a) Suppose the process ID (PID) is 5000. Assume that each new child process will have the next sequential PID that is available (5001, 5002, etc.). Please write down the output sequences of these processes!

b) What happen if we delete line 12, 14, 16, and 18 [sleep()] ?

c) What happen if we delete line 24 and 25 [waitpid(-1, NULL, 0)]?

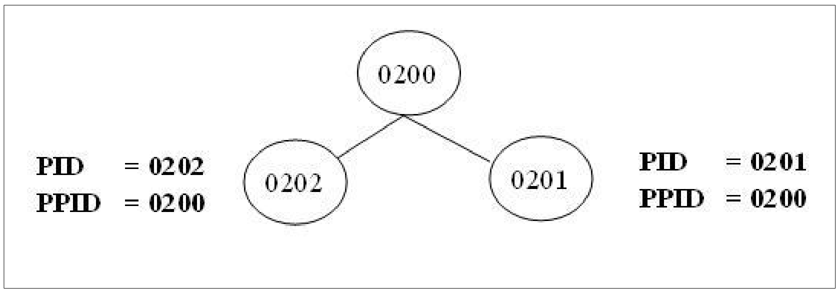
Fork V

Please write down the output of program "fork6"! Assume the first PID is 5000.

```
/* fork6.c (c) 2008 Rahmat M. Samik-Ibrahim  v.08.11.04.00
 * getpid() = get the current PID (Process ID).
 * fork()   = creates a new process by duplicating.
 * wait()   = wait until one of its children terminates.
 * GFDLike License */

#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#define  STRING1 "PID[%5.5d] starts.\n"
#define  STRING2 "PID[%5.5d] passes.\n"
#define  STRING3 "PID[%5.5d] terminates.\n"

main()
{
   printf(STRING1, (int) getpid());
   if (fork() == 0)
      fork();
   wait(NULL);
   fork();
   wait(NULL);
   printf(STRING2, (int) getpid());
   wait(NULL);
   printf(STRING3, (int) getpid());
}
```

Fork VI

One process has one PPID. PPID stands for the Parent Process ID. Below are the relation between PID and PPID:



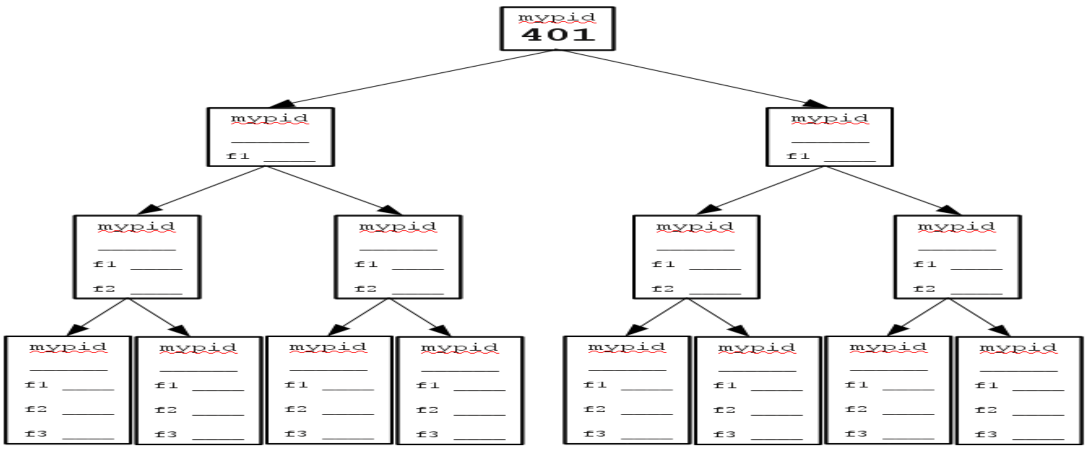
Look at the following C program below. If the initial PID of this program is 0401 and the PPID of 0401 is 0400, print the OUTPUT of the program.

```
//Forkloop.c
//getpid()  : return the Process ID (PID)
//getppid() : return the Parent Process ID (PPID)

#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>
#define STR "PID = %4.4d, PPID = %4.4d, Val = %4.4d\n"



int main()
{
      int count=0, loop=3, val=4;
      while (count != loop)
      {
            if(fork()>0) val--;
            wait(NULL);
            count++;
      }

      printf(STR, getpid(), getppid(), val);
}
```

Fork VII (2009)

```
/* triplefork.c (c) 2009 Rahmat M. Samik-Ibrahim, GPL-like */
/* ************ ******************************************* */
#include <sys/types.h>
#include <sys/wait.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#define DISPLAY \
"mypid[%3.3d] - f1[%3.3d] - f2[%3.3d] -  f3[%3.3d]\n"
/***************************************** main ** */
main(void) {
   int f1, f2, f3, mypid;
   f1 = (int) fork();
   sleep(1);
   f2 = (int) fork();
   sleep(2);
   f3 = (int) fork();
   sleep(3);
   mypid=(int) getpid();
   printf(DISPLAY, mypid, f1, f2, f3);
   waitpid(-1,NULL,0);
   waitpid(-1,NULL,0);
   waitpid(-1,NULL,0);
   exit (0);
}
```

Please write down the tree while matching the program output of "triplefork" (the first mypid is 401). Remember that these are concurrent processes! **DO NOT ASSUME ANY PROCESS SEQUENCE!**



```
PROGRAM OUTPUT
mypid[____________] - f1[402] - f2[000] -  f3[408]
mypid[____________] - f1[402] - f2[000] -  f3[000]
mypid[____________] - f1[000] - f2[000] -  f3[407]
mypid[____________] - f1[000] - f2[000] -  f3[000]
mypid[____________] - f1[000] - f2[404] -  f3[406]
mypid[____________] - f1[000] - f2[404] -  f3[000]
mypid[____________] - f1[402] - f2[403] -  f3[405]
mypid[____________] - f1[402] - f2[403] -  f3[000]
```

MultiThreads

```java
// MultiThreads (c)2006 Rahmat M. Samik-Ibrahim, GPL-like //
// ************************************ MultiThreads ***   //
public class MultiThreads {
   public static void main(String args[]) {
      Engine   engine = new Engine(THREAD_COUNT);
      Thread[] player = new Thread[THREAD_COUNT];
      for (int ii=0; ii<THREAD_COUNT ; ii++) {
         player[ii]   = new Thread(new Player(ii,engine));
         player[ii].start();
      }
   }
   private static final int THREAD_COUNT = 4;
}
// ****************************************** Player *** //
class Player implements Runnable {
   Player(int count, Engine eng) {
      engine       = eng;
      player_count = count;
   }
   public void run() {
      engine.play(player_count);
   }
   private  Engine engine;
   private  int    player_count;
}
// ****************************************** Engine *** //
class Engine {
   public Engine(int count) {
      idx     = count-1;
      control = new Semaphore[count];
      for (int ii=0; ii<count ; ii++) {
         control[ii] = new Semaphore();
      }
   }
   public void play(int ii) {
      if (ii < idx)  {
         control[ii+1].acquire();
      }
      System.out.println("Player " + ii + " is up...");
      control[ii].release();
   }
   private  int         idx;
   private  Semaphore[] control;
}
// *************************************** Semaphore *** //
class Semaphore {
   public Semaphore() {
      value = 0;
   }
   public synchronized void acquire() {
      while (value == 0) {
         try   { wait(); }
         catch (InterruptedException e) { }
      }
      value--;
   }
   public synchronized void release() {
      value++;
      notify();
   }
```

```
    private int value;
 }
```

a) Please write down the output of this java program!
b) Please explain briefly, the purpose of using the semaphores in this java program!
c) Please, slightly modify the "Engine class" so that the output sequence will be the opposite of point (a). *(Hint: 3 lines only, lah! :-)*.

Synchronization I

a) How many semaphore objects are used in this following Java program? Name them one by one!
b) Write down the output of the Java program!

```
/*******************************************************/
/* Sakit (c)2007 Rahmat M. Samik-Ibrahim, GPL-like       */
/*******************************************************/

public class Sakit {
   public static void main(String args[]) {
      Engine    engine = new Engine(strings, strseq);
      Thread[] printer = new Thread[strings.length];
      for (int ii = 0; ii < strings.length; ii++) {
         printer[ii]=new Thread(new Printer(ii, engine));
         printer[ii].start();
      }
   }
   private final static String strings[]=
      {"Bapak", "Budi", "kepala", "si", "sakit"};
   private final static int strseq[]=      {0,3,1,4,2};
}
class Engine {
   Engine(String str[],int strseq[]) {
      this.str    = str;
      this.strseq = strseq;
      semaphore   = new Semaphore[str.length];
      for (int ii=0; ii<str.length; ii++) {
         semaphore[ii] = new Semaphore();
      }
      sequence    = 0;
      semaphore[strseq[sequence++]].release();
   }
   public void go(int ii) {
      semaphore[ii].acquire();
      System.out.print(str[ii] + " ");
      if (sequence < strseq.length)
         semaphore[strseq[sequence++]].release();
      else
         System.out.println();
   }
   private Semaphore[] semaphore;
   private String      str[];
   private int         strseq[];
   private int         sequence;
}
class Printer implements Runnable {
   Printer(int ii, Engine ee) {
      number = ii;
      engine = ee;
   }
   public void run() {
      engine.go(number);
   }

   private int    number;
   private Engine engine;
}

/**********************************************************/
class Semaphore {
   public Semaphore()      { value = 0; }
   public Semaphore(int v) { value = v; }
   public synchronized void acquire() {
      while (value == 0) {
         try   { wait(); }
         catch (InterruptedException e) { }
      }
      value--;
   }
   public synchronized void release() {
      value++;
      notify();
   }
   private int value;
}
```

Synchronization II

```
/**************************************************************/
/* MultiStrings (c) 2008 Rahmat M. Samik-Ibrahim, GPL-like    */
/* $Date: 2008/06/25 12:12:30 $ $Revision: 1.1 $            */
/**************************************************************/

public class MultiStrings {
   public static void main(String args[]) {
      Engine    engine = new Engine(strings, strseq);
      Thread[] printer = new Thread[strings.length];

      for (int ii = 0; ii < strings.length; ii++) {
         printer[ii]=new Thread(new Printer(ii, engine));
         printer[ii].start();
      }
   }
   private final static String strings[]= { "a", "an", "and",
      "as", "Design", "Extended", "Implementation", "Machine",
      "Manager", "Operating", "Resource", "Systems", "The"};
   private final static int string_array[][] =
      {{12,9,11,3,1,5,7}, {12,9,11,3,0,10,8}, {9,11,4,2,6}};
   /*  NameA=0      NameC=1   NameE=2
       NameB=0      NameD=1   NameF=2 */
   private final static int strseq[] = string_array[VALUE];
}

/*
/**************************************************************/
class Engine {
   Engine(String str[],int strseq[]) {
      this.str    = str;
      this.strseq = strseq;
      semaphore   = new Semaphore[str.length];
      for (int ii=0; ii<str.length; ii++) {
         semaphore[ii] = new Semaphore();
      }
34       display     = true;
35       sequence    = 0;
36       semaphore[strseq[sequence++]].release();
37    }
38    public void go(int ii) {
39       semaphore[ii].acquire();
40       if (display) {
41          System.out.print(str[ii] + " ");
42          if (sequence < strseq.length) {
43             semaphore[strseq[sequence++]].release();
44          } else {
45             System.out.println();
46             display     = false;
47             for (int jj=0;jj<str.length;jj++) {
48                semaphore[jj].release();
49             }
50          }
51       }
52    }
53    private Semaphore[] semaphore;
54    private String      str[];
55    private int         strseq[];
56    private int         sequence;
57    private boolean     display;
58 }
60 /*************************************************************/
61 class Printer implements Runnable {
62    Printer(int ii, Engine ee) {
63       number = ii;
64       engine = ee;
65    }
66    public void run() {
67       engine.go(number);
68    }
69    private int    number;
70    private Engine engine;
71 }
73 /*************************************************************/
74 class Semaphore {
75    public Semaphore()      { value = 0; }
76    public Semaphore(int v) { value = v; }
77    public synchronized void acquire() {
78       while (value == 0) {
79          try   { wait(); }
80          catch (InterruptedException e) { }
81       }
82       value--;
83    }
84    public synchronized void release() {
85       value++;
86       notify();
87    }
88    private int value;
89 }
90 /*************************************************************/
```

a) See line 19 for the "**VALUE**" of line 21. What is the output of this program?
b) What is the value of "strseq.length" in line 42?
c) What is the value of "str.length" in line 47?
d) What is the purpose of the loop in line (47 to 49)? What happen if we delete those lines?

Synchronization III

```java
// (c) 2003 Silberschatz, Galvin and Gagne.
// Slightly modified by Rahmat M. Samik-Ibrahim

import java.util.*;

public class ProducerConsumer
{
   public static void main(String args[]) {
      BoundedBuffer server  = new BoundedBuffer();
      Thread producerThread = new Thread(new Producer(server));
      Thread consumerThread = new Thread(new Consumer(server));

      producerThread.start();
      consumerThread.start();
   }
}

class Producer implements Runnable
{
   private int               prod_time;
   private BoundedBuffer     buffer;
   private static final int  DEFAULT_TIME = 3;

   public Producer(BoundedBuffer bf)           { init(bf, DEFAULT_TIME); }
   public Producer(BoundedBuffer bf, int time) { init(bf, time); }

   private void init(BoundedBuffer bf, int time) {
      buffer    = bf;
      prod_time = time;
   }

   public void run() {
      Date message;
      while (true) {
         System.out.println("Producer processing");
         DelayUtilities.process_time(prod_time);
         message = new Date();
         System.out.println("Producer produced " + message);
         buffer.insert(message);
      }
   }
}

class Consumer implements Runnable
{
   private BoundedBuffer     buffer;
   private int               cons_time;
   private static final int  DEFAULT_TIME = 3;

   public Consumer(BoundedBuffer bf)           { init(bf, DEFAULT_TIME); }
   public Consumer(BoundedBuffer bf, int time) { init(bf, time);         }

   public void run() {
      Date message;
      while (true) {
         System.out.println("Consumer processing");
         DelayUtilities.process_time(cons_time);
         System.out.println("Consumer wants to consume.");
         message = (Date)buffer.remove();
      }
   }

   private void init(BoundedBuffer bf, int time) {
      buffer    = bf;
      cons_time = time;
   }
}

class BoundedBuffer
{
   private static final int  DEFAULT_SIZE = 2;
   private Semaphore         mutex, empty, full;
   private int               count, in, out, buffer_size;
   private Object[]          buffer;

   public BoundedBuffer(int bfsize) { init(bfsize);       }
   public BoundedBuffer()           { init(DEFAULT_SIZE); }

   public void insert(Object item) {
      empty.acquire();
      mutex.acquire();
      ++count;
      buffer[in] = item;
      in = (in + 1) % buffer_size;
      System.out.print("INSERT: " + item);
      if (count == buffer_size)
         System.out.println(" Buffer Size (FULL)");
      else
         System.out.println(" Buffer Size (" + count + ")");
      mutex.release();
      full.release();
   }
   public Object remove() {
      full.acquire();
      mutex.acquire();
      --count;
      Object item = buffer[out];
      out = (out + 1) % buffer_size;
      System.out.print("REMOVE: " + item);
      if (count == 0)
         System.out.println(" Buffer Size (EMPTY)");
      else
         System.out.println(" Buffer Size (" + count + ")");
      mutex.release();
      empty.release();
      return item;
   }

   private void init(int bfsize) {
      count       = 0;
      in          = 0;
      out         = 0;
      buffer_size = bfsize;
      buffer      = new Object[bfsize];
      mutex       = new Semaphore(1);
      empty       = new Semaphore(bfsize);
      full        = new Semaphore(0);
   }
}

class Semaphore
{
   private int value;

   public Semaphore(int value) { this.value = value; }

   public synchronized void acquire() {
      while (value <= 0) {
         try { wait(); }
         catch (InterruptedException e) { }
      }
      value--;
   }

   public synchronized void release() {
      ++value;
      notify();
   }
}


class DelayUtilities
{
   public static void process_time(){ process_time(PROCESS_TIME); }
   public static void process_time(int duration) {
      int sleeptime = (int) (duration * Math.random() );
      try   { Thread.sleep(sleeptime*1000); }
      catch (InterruptedException e) {}
   }
   private static final int PROCESS_TIME = 5;
}
```

Analyze this ProducerConsumer.java program.

a) Modify class ProducerConsumer (ONLY) so that these following variables can be easily assigned: “Bounded Buffer Size (BBS)”, “Producer Process Time (PPT)”, “Consumer Process Time (CPT)”. For example, BBS=3, PPT=4; CPT=5. Do not re-write the class, just explain the changes!

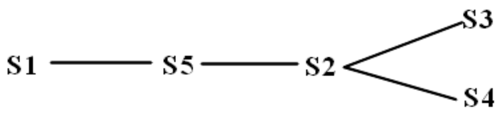
b) Replace class “ProducerConsumer” with “ProducerDistributorConsumer” with two buffers and add a new class “Distributor”. The first buffer is between Producer and Distributor and the second class is between Distributor and Consumer.

Synchronization IV

There exist four processes P1(S1), P2(S2, S3), P3(S4), P4(S5), where Pn(Sx) means Pn=process number, Sx=Statement x belongs to Pn.

CPU will execute the process statements based on the following order :

1. CPU executes S1
2. CPU executes S5
3. CPU executes S2
4. CPU may execute S3, S4 or both at the same time



Note: when CPU executes S1 of P1, other process will wait. After executing S1, CPU executes S5 of P4 and so on .

To achieve those execution order, write the solution code by using **wait(S)** and **signal(S)** method. Determine how many semaphore objects you will use. Implement your code on the empty space below:

//initialize your semaphore objects here			
1	**P2**	**P3**	**P4**
//**P1 codes**	//**P2 codes**	//**P3 codes**	//**P4 codes**

Deadlock

There exist four processes in the system. Total resources in the system are P(13), Q(19), R(15).
The process sequence : <P2, P4, P5, P1, P3>
By using deadlock avoidance, please specify :

a) The need table
b) Is the system safe or not ? Prove it!
c) What happen to the system if P3 ask one more resource R ? is it safe or not ?

Processes	*Allocation*			*Maximum*			*Available*		
	P	Q	R	P	Q	R	P	Q	R
P1	2	2	1	11	16	14	2	7	3
P2	3	2	1	4	8	3			
P3	2	2	2	5	7	6			
P4	4	5	6	9	11	10			
P5	0	1	2	3	2	4			

Memory I

Explain briefly these following terms:

a) logical address
b) demand paging
c) page fault
d) reference string
e) copy on write

Memory II

A system has specifications :

- Total page in logical memory: 8
- Total frame in physical memory: 16
- Page size: 128 byte
- Number of process: 16
- One Page Table Entry (PTE) size: 2 byte
- Measurement: 1kb= 210 byte

A page table entry has the following format: (page number, frame number). The entry list of page table:

(0,0), (1,8), (2,11), (3, 6), (4, 15), (5, 1), (6, 7), (7, 10)

a) Determine the last address of logical memory! (the address starts from 0)

b) Determine the last address of physical memory! (the address starts from 0)
c) Determine the physical address for logical address 800!
d) Determine the logical address for physical address 2000!
e) How many bytes needed to put the entire processes' page table in physical memory?

Memory III

A system has specification:

- Page replacement algorithm: Least-recently-used (LRU)
- Available frame: 2
- Reference string : 0, 1, 3, 4, 5, 5, 8, 0, 3, 3

a) How many page faults exist?
b) If LRU is implemented using stack, write the stack value for each access on reference string.

c) How many page faults exist if the system uses FIFO algorithm?

Memory IV

A system has specification:

- Total available physical memory frame : 300
- Total processes : 6
- The frame needed by each process has the following format (Process ID, Total frame).
- List of needs :

(0, 40), (1, 60), (2, 100), (3, 20), (4, 80), (5, 100)
Each process requires some frames to run. Unfortunately the total available frames are limited. The system can not supply all the requested frames to every process. There are two simple techniques to solve this problem, **Equal Allocation** and **Proportional Allocation** .

a) Determine total frames given to each process if the system uses equal allocation mechanism!
b) Determine total frames given to each process if the system uses proportional allocation algorithm!

Memory V

A system uses demand paging technique to increase the efficiency of memory utilization. In 1000 time access, 400 page faults occur. Determine the **Effective Access Time (in nanoseconds)** of Demand Paging if: memory access time = 100ns and page-fault time = 8ms.

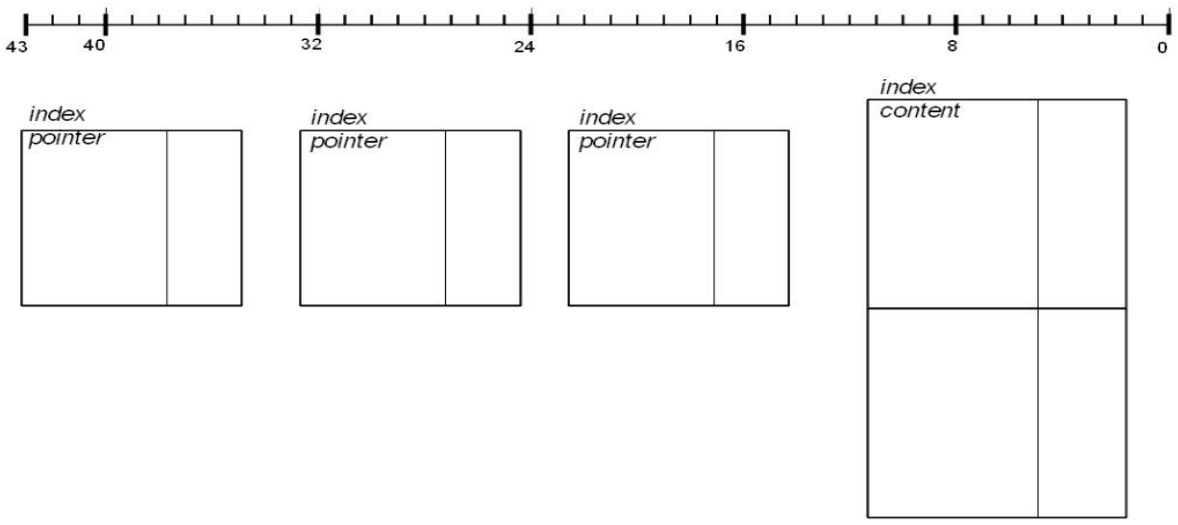
Linux Three-Level Page Table I

This following, **008 0200 8004** (HEX), is a valid 43 bit Linux Virtual Address with three level page tables: Global Directory (10 bits), Page Middle Directory (10 bits), and Page Table (10 bits).

a) Convert the base-16 address above into base-2.

b) Complete the following diagram with its table names, indexes (in base-16), pointers (in arrow form), and memory contents (whatever/random). You may use **dotes** "**. . .**" for "**and so on**".

c) What is the size of a memory frame?



Linux Three-Level Page Table II

What if the address is **004 0100 4002** (HEX)?

Linux Three-Level Page Table III

What if the address is **000 0000 0000** (HEX) ?

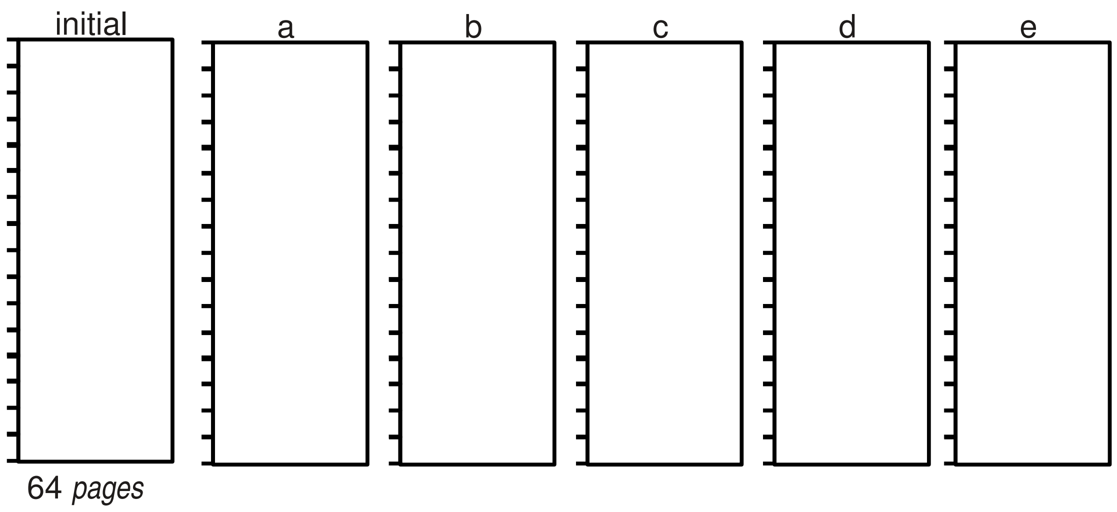
Linux Three-Level Page Table IV (2009)

What if the address is **010 1010 1010** (HEX) ?

Buddy Algorithm I

Basically, the "Buddy Algorithm" allocates pages in the power-of-2. The request will be rounded up to the next highest power of 2. Give a simple illustration of the this algorithm. Suppose, there exists a single contiguous memory of 64 pages.

(a) Process A requests 7 pages. (b) Process B requests 3 pages. (c) Process C requests 9 pages. (d) Process B returns its request. (e) Process D requests 9 pages.



Buddy Algorithm II

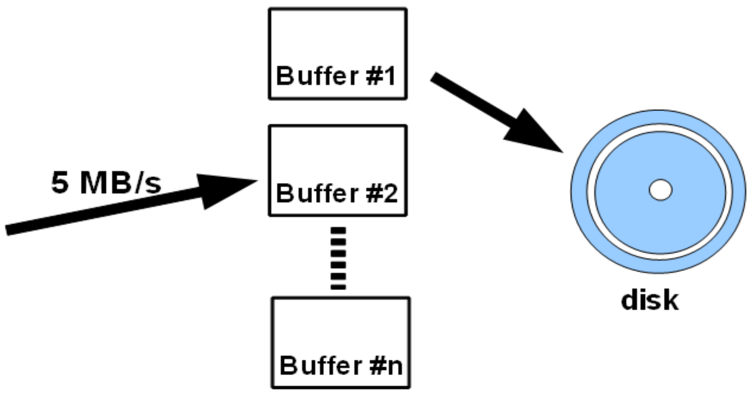
(See picture above). (a) Process A requests 9 pages. (b) Process B requests 7 pages. (c) Process C requests 3 pages. (d) Process B returns its request. (e) Process D requests 1 page.

Buddy Algorithm III

(See picture above). (a) Process A requests 5 pages; (b) Process B requests 3 pages; (c) Process C requests 5 pages; (d) Process B returns its request; (e) Process D requests 2 pages.

HardDisk (I/O) I

- The disk rotates at 6000 RPM. Each track holds 1000 sectors @ 10 kbytes.
- Whenever one of these two buffers (@10 kbytes) is empty, the system will refill it at a constant rate of 5 Mbytes per second.
- At t=0, the disk head is at sector=0, Buffer #1 is full, and Buffer #2 is empty.



a) **BEST CASE:** For a maximal effective transfer rate, at least how many buffers are needed? How much will be that effective transfer rate? Explain!

b) **WORST CASE**: For a maximal effective transfer rate, at least how many buffers are needed? How much will be that effective transfer rate? Explain!

HardDisk II

A disk with the following specifications:

- Using five (5) platters
- One platter consists of two (2) surfaces (top surface #0 and bottom surface #1).
- One (1) surface capacity: 5 Gb
- Total tracks on one (1) surface: 2500 (cylinder #0 - #2499).
- Speed rotation: 6000 rpm.
- Total sectors in one track: 500 (sector #0 - #499).
- Time needed to move from one track to adjacent track: 1 ms (for example: moving from track 1 to track 2 = 1ms, from track 0 to track 2 = 2ms).
- Assume, only one head active reading/writing. Time needed to move from surface #0 to surface #1 is 0 ms.
- Disk scheduling algoritm: First Come First Served.
- At T=0, the head position is above cylinder #0, sector #0.
- Measurement: 1 kbyte = 1000 byte; 1 Mbyte = 1000 kbyte; 1 Gbyte = 1000 Mbyte.

Questions:

a) Determine the disk size.

b) Determine the capacity of one cylinder.

c) How long it takes to read/write one sector?

d) Determine the time (mS) to move from surface#0, track#0, sector#0 to surface#0, track#4,sector #399 ([0,0,0] →[0,4,399]).

e) Determine the time (mS) to move from [0,0,0] → [0,0,499] → [0,3,99] → [0,3,499] → [0,2,249].

HardDisk III

A disk consists of 5000 cylinders (from cylinder 0 – cylinder 4999). Some processes request to read/write some cylinders. System pools the entire request on the following queue:

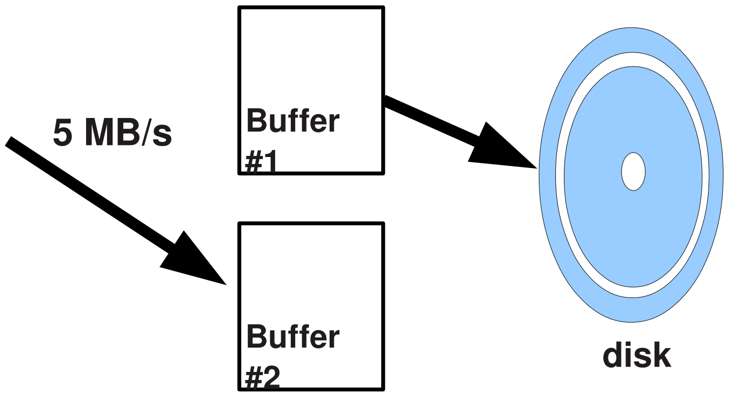
Queue(Q)=200, 1500, 3500, 2000, 4000, 500

Right now the disk is reading cylinder 3000. Determine the cylinder access sequence and its total movement if the following disk scheduling algorithms are used:

a) SSTF

b) LOOK (assume the head moves to the left direction)

HardDisk IV (2009)



- The disk rotates at 6000 RPM. Each track holds 1000 sectors @ 10 kbytes.
- Whenever one of these two buffers (@10 kbytes) is empty, the system will refill it at a constant rate of 5 Mbytes per second.
- At t=0, the disk head is at sector=0, Buffer #1 is full, and Buffer #2 is empty.

a) How long will it take to write down 1Mbytes on the same track starting sector 0, sector 1, and so on?

b) How long will it take to write down 1Mbytes on the same track starting sector 999, sector 998, sector 997, and so on?

Disk Partitions I

```
Select device       ----first---- --geom/last-- ------sectors-----
  Device             Cyl Head Sec  Cyl Head Sec  Base  Size      Kb
  /dev/c0d1                         32   16  63
                       0    0   0   31   15  62     0 32256   16128
Num Sort   Type
 0   p0   81 MINIX    __   __  __   __   __  __    63 _____   _____
 1   p1   81 MINIX    __   __  __   __   __  __  ____ _____   _____
 2   p2   81 MINIX    __   __  __   __   __  __  ____ _____   _____
 3   p3   81 MINIX    __   __  __   __   __  __  ____ _____   _____
```

a) Divide a disk into four (4) main partitions. The first partition size is 2048 kbytes. the second one is 4096 kbytes, and the third one is 8192 kbytes. Please fill the blanks of the scheme above.

b) Why does the first partition not start from track #0?

Disk Partitions II

Please fill the blanks of this following scheme:

```
 Select device       ----first---- --geom/last-- ------sectors-----
  Device             Cyl Head Sec  Cyl Head Sec  Base  Size      Kb
  /dev/c0d1                         65   16  63
                       0    0   0   64   15  62     0  65520   32760
Num Sort   Type
 0   p0  81 MINIX      0    1   0   __   __  __    63  _____   _____
 1   p1  81 MINIX      4    3   0   __   __  __  4221  _____   _____
 2   p2  81 MINIX     12    5   0   __   __  __ 12411  _____   _____
 3   p3  81 MINIX     28    9   0   __   __  __ 28971  _____   _____
```

Disk Partitions III

```
Select device      ----first----   --geom/last-- ------sectors-----
  Device             Cyl Head Sec    Cyl Head Sec  Base  Size      Kb
  /dev/c0d3                           65   16  63
                       0    0   0     64   15  62     0 65520   32760
Num Sort   Type
 0   p0  81 MINIX    ___  ___  ___   ___  ___ ___    63 _____    8032
 1   p1  81 MINIX    ___  ___  ___   ___  ___ ___  ____ _____   16632
 2   p2  81 MINIX    ___  ___  ___   ___  ___ ___  ____ _____    4032
 3   p3  81 MINIX    ___  ___  ___   ___  ___ ___  ____ _____   _____
```

```
Select device      ----first----   --geom/last-- ------sectors-----
  Device             Cyl Head Sec    Cyl Head Sec  Base  Size      Kb
  /dev/c0d3p1                         65   16  63
                      16    0   0     48   15  62 16128 33264   16632
Num Sort   Type
 0    0  81 MINIX    ___  ___  ___   ___  ___ ___  ____ _____    4032
 1    1  81 MINIX    ___  ___  ___   ___  ___ ___  ____ _____    4032
 2    2  81 MINIX    ___  ___  ___   ___  ___ ___  ____ _____    4032
 3    3  81 MINIX    ___  ___  ___   ___  ___ ___  ____ _____   _____
```

a) Please fill the blanks of the above scheme.
b) What is the size of partition /dev/c0d3p2?
c) What is the size of partition /dev/c0d3p1s2?

Disk Partitions IV (2009)

a) Please fill the blanks of this following scheme.

```
Select device       ----first----   --geom/last-- ------sectors-----
  Device             Cyl Head Sec   Cyl Head Sec  Base  Size      Kb
  /dev/c0d3                           406   16  63
                       0    0   0   405    15   62     0  409248 204624
Num Sort   Type
 0   p0  81 MINIX      0    1   1   ___  ___  ___   64   _____   50368
 1   p1  81 MINIX    100    0   0   ___  ___  ___  ____  _____   50400
 2   p2  81 MINIX    200    0   0   ___  ___  ___  ____  _____   50400
 3   p3  81 MINIX    300    0   0   ___  ___  ___  ____  _____   _____
```

```
Select device       ----first----   --geom/last-- ------sectors-----
  Device             Cyl Head Sec   Cyl Head Sec  Base   Size      Kb
  /dev/c0d3                         406  16  63
  /dev/c0d3:1        100    0   0   199  15  62  100800 100800  50400
Num Sort   Type
 0    0  81 MINIX    100    1   1   ___ ___ ___  100864  _____  12568
 1    1  81 MINIX    125    0   0   ___ ___ ___  ______  _____  12600
 2    2  81 MINIX    150    0   0   ___ ___ ___  ______  _____  12600
 3    3  81 MINIX    175    0   0   ___ ___ ___  ______  _____  _____
```

b) What is the size of partition /dev/c0d3p2?

c) What is the size of partition /dev/c0d3p1s2?

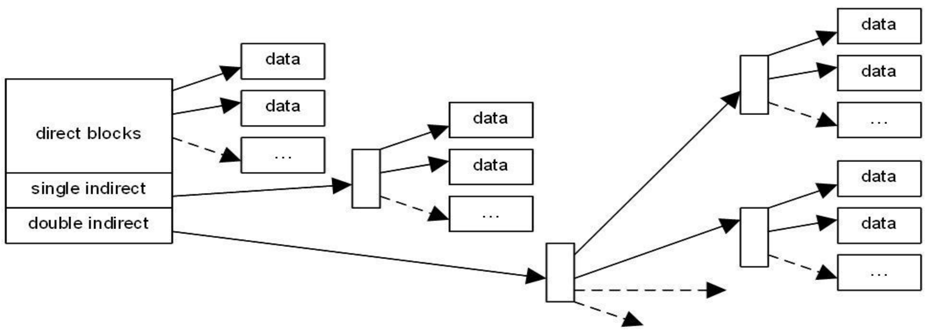
File System I

This file system is using an inode (unix) alike allocation method. The pointer size is 4 bytes. Supposed there are **12** pointers in the i-node. The first **10** ones point to "**direct blocks**", i.e. the content (data) of the file. The next **one** points to a single "**single indirect block**", which points to "**direct blocks**". The last **one** points to a single "**double indirect block**", which points to "**single indirect blocks**".

a) If the block size is 100 bytes, what will be the maximum size of the file?

b) If the block size is 1000 bytes, what will be the maximum size of the file?

c) If the block size is N bytes, what will be the maximum size of the file?



File System II

a) (See picture above). This file system is using an inode (unix) alike allocation method. The pointer size is **2** bytes and the block size is **1000** bytes. Supposed there are **12** pointers in the i-node. The first **10** ones point to "**direct blocks**", ie. the content (data) of the file. The next one points to a single "**single indirect block**", which points to "**direct blocks**". The last **one** points to a single "**double indirect block**", which points to "**single indirect blocks**". What is the maximum size of a file?

b) What is the maximum size of a file if the pointer size is **PS** bytes, the block size is **BS** bytes, and there are **PTR** pointers in the i-node?

True/False

1. T / F An Operating System is a program (2010).
2. T / F An Operating System is an intermediary between the users and the hardware of a computer (2010).
3. T / F An Operating System is a license (2010).
4. T / F A System Program is a part of the kernel (2010).
5. T / F A System Program is also called an Applications Program (2010).
6. T / F A System Program is a also called System Call (2010).
7. T / F The VMware family (eg. "***VMware Player***") is a MINIX tool (2010).
8. T / F The VMware family (eg. "***VMware Player***") runs as an application on top of a host operating system (2010).
9. T / F The VMware family (eg. "***VMware Player***") does not require virtualization support from the host kernel (2010)

Integer Arithmetics (10-12)(2010):

```
int  ii = 0xdead;
int  jj = 0xface;
```

10. T / F `ii + jj = 0x1d97b;`
11. T / F `ii + jj = 0x11101100101111011;`
12. T / F `ii + jj = 0xdeadface;`
13. T / F Creating a new ***thread*** is much faster then creating a new process (2010).
14. T / F Creating a new ***thread*** is much slower then creating a new process (2010).
15. T / F Creating a new ***thread*** is as fast as creating a new process (2010).
16. T / F For the `fork()` system call, the return code for the `fork()` of child process is 0 (2010).
17. T / F For the `fork()` system call, the return code for the `fork()` of parent process is 0 (2010).
18. T / F For the `fork()` system call, the process ID of the child process is 0 (2010).
19. T / F A Round Robin scheduling system, with a large time quantum/slice is less responsive (2010).
20. T / F A Round Robin scheduling system, with a large time quantum/slice is more responsive (2010).
21. T / F A Round Robin scheduling system, can not be a non-preemptive system (2010).
22. T / F The memory page fault frequency can be reduced if most processes are CPU bound (2010).
23. T / F The memory page fault frequency can be reduced if enlarging the page size (2010).
24. T / F The memory page fault frequency can be reduced if most processes are I/O bound (2010).
25. T / F There size of a disk with 5 double side platters, 5000 cylinders, 1024 sectors per track, and 1 kBytes per sector is > 60 Gbytes (2010).
26. T / F There size of a disk with 5 double side platters, 5000 cylinders, 1024 sectors per

track, and 1 kBytes per sector is > 40 Gbytes (2010).

27. T / F There size of a disk with 5 double side platters, 5000 cylinders, 1024 sectors per track, and 1 kBytes per sector is > 20 Gbytes (2010).
28. T / F An address generated by the CPU is commonly referred to as a register address (2010).
29. T / F An address generated by the CPU is commonly referred to as a logical address (2010).
30. T / F An address generated by the CPU is commonly referred to as a physical address (2010).
31. T / F A strategy to load pages only as there are needed is known as shared paging (2010).
32. T / F A strategy to load pages only as there are needed is known as swap paging (2010).
33. T / F A strategy to load pages only as there are needed is known as demand paging (2010).
34. T / F The SSTF (Shortest Seek Time First) disk scheduling algorithm, may cause starvation for some request (2010).
35. T / F The SSTF (Shortest Seek Time First) disk scheduling algorithm, is the most optimal scheduling algorithm (2010).
36. T / F The SSTF (Shortest Seek Time First) disk scheduling algorithm, selects the request with the least seek time from the current head position (2010).
37. T / F When the disk arm moves from end to end, servicing all requests in its path, the scheduling mechanism is known as Shortest Seek Time First (SSTF) (2010).
38. T / F When the disk arm moves from end to end, servicing all requests in its path, the scheduling mechanism is known as SCAN (2010).
39. T / F When the disk arm moves from end to end, servicing all requests in its path, the scheduling mechanism is known as LOOK (2010).
40. T / F A Real Time System may have these following characteristic: general/ multipurpose (2010).
41. T / F A Real Time System may have these following characteristic: large size (2010).
42. T / F A Real Time System may have these following characteristic: real number only, no integer (2010).
43. T / F A Multimedia System may have these following characteristic: large files (2010).
44. T / F A Multimedia System may have these following characteristic: sensitive to timing delay (2010).
45. T / F A Multimedia System may have these following characteristic: high data rates (2010).
46. T / F A spin-lock semaphore, wastes CPU cycles (2010).
47. T / F A spin-lock semaphore, is also called a busy-waiting semaphore (2010).
48. T / F A spin-lock semaphore, is less accurate compared to the block/wakeup method (2010).
49. T / F Most General Purpose Operating Systems provide "Deadlock Prevention" mechanism (2010).
50. T / F Most General Purpose Operating Systems provide "Deadlock Avoidance" mechanism (2010).

51. T / F Most General Purpose Operating Systems provide "Nothing" to avoid Deadlock (2010).
52. T / F The page fault frequency can be reduced if most processes are CPU bound (2010).
53. T / F The page fault frequency can be reduced if enlarging the page size (2010).
54. T / F The page fault frequency can be reduced if most processes are I/O bound (2010).
55. T / F A Direct Memory Access (DMA) controller increases I/O interrupts (2010).
56. T / F A Direct Memory Access (DMA) controller reduces I/O interrupts (2010).
57. T / F A Direct Memory Access (DMA) controller has nothing to do with I/O interrupts (2010).
58. T / F In a Memory Mapped I/O system, the I/O instructions are special (2010).
59. T / F In a Memory Mapped I/O system, a certain memory address range is reserved for the I/O interfaces (2010).
60. T / F In a Memory Mapped I/O system, the I/O uses the same instructions as the Memory References (2010).
61. T / F The Translation Look-aside Buffer (TLB), is a software strategy of a page-table implementation (2010).
62. T / F The Translation Look-aside Buffer (TLB), is a small fast look-up cache (2010).
63. T / F The Translation Look-aside Buffer (TLB), should have the entry size as large as the page-table size (2010).

64. T / F A strategy to load pages only as there are needed is known as shared paging (2010).
65. T / F A strategy to load pages only as there are needed is known as swap paging (2010).
66. T / F A strategy to load pages only as there are needed is known as demand paging (2010).
67. T / F The host uses Interrupts to handle busy-waiting/ polling events (2010).
68. T / F The host uses Interrupts to handle asynchronous events (2010).
69. T / F The host uses Interrupts to handle both maskable and non-maskable events (2010).
70. T / F In a 32-bit system, the external address size is at most 32 bits (2009).
71. T / F In a 32-bit system, the ALU size is at most 32 bits (2009).
72. T / F In a 32-bit system, the external data bus size is at most 32 bits (2009).
73. T / F In a 32-bit system, the register integer size is at most 4 bytes (2009).
74. T / F A data cache is a collection of data duplicating values that were stored elsewhere, earlier (2009).
75. T / F A data cache is expected miss rate declines with an increased the cache size (2009).
76. T / F A data cache is fetch time is much shorter compared if it is fetched from the original place (2009).
77. T / F A data cache is expected hit rate should be close to 1 (2009).
78. T / F In a write-through cache, every write to the cache causes a synchronous write to the backing store (2009).
79. T / F In a write-through cache, is also called a write-back cache (2009).

80. T / F In a write-through cache, is also called a delayed-write cache (2009).
81. T / F The address binding of instructions and data to memory address can be done at compile time (2009).
82. T / F The address binding of instructions and data to memory address can be done at load time (2009).
83. T / F The address binding of instructions and data to memory address can be done at execution time (2009).

Thread T1 and T2 are sharing integers “ii” and “jj”. Suppose semaphore functions are implemented with “wait(): decreases the semaphore integer” and “signal(): increases the semaphore integer”. Initially, S1=1 and S2=0. Which one of the following will have no race-condition (84-86):

84. T / F

```
    T1               T2
wait(S1)         wait(S1)
ii = ii + 1      ii = ii - 1
signal(S1)       signal(S1)
wait(S2)         signal(S2)
jj = ii + jj
```

85. T / F

```
     T1          T2
wait(S1)         wait(S2)
ii = ii + 1      ii = ii - 1
jj = ii + jj     signal(S2)
signal(S1)
```

86. T / F

```
     T1          T2
  wait(S1)       wait(S1)
  ii = ii + 1    ii = ii - 1
  signal(S1)     signal(S1)
  wait(S2)
  jj = ii + jj
  signal(S2)
```

87. T / F Dynamic Linking Libraries are combined into the binary image together with all other modules (2009).
88. T / F Dynamic Linking Libraries are the linking is postponed until executing time (2009).
89. T / F Dynamic Linking Libraries are uses “.DLL” extension in Microsoft Windows family (2009).
90. T / F The total time to prepare a disk system to read or write a data sector is transmission time (2009).
91. T / F The total time to prepare a disk system to read or write a data sector is latency time (2009).

92. T / F The total time to prepare a disk system to read or write a data sector is seek time (2009).

93. T / F Copy-on-write is a common technique used by several operating systems, including the Microsoft Windows Family (NT, 2000, XP, Vista, etc) (2009).

94. T / F Copy-on-write is a common technique used by several operating systems, including Linux (2009).

95. T / F Copy-on-write is a common technique used by several operating systems, including Solaris (2009).

96. T / F The FIFO algorithm is a strategy to manage free memory for kernel processes (2009).

97. T / F The buddy system is a strategy to manage free memory for kernel processes (2009).

98. T / F The slab allocation is a strategy to manage free memory for kernel processes (2009).

99. T / F When the disk arm moves from end to end, servicing all requests in its path, the scheduling mechanism is known as First Come First Served (FCFS) (2009).

100. T / F When the disk arm moves from end to end, servicing all requests in its path, the scheduling mechanism is known as Shortest Seek Time First (SSTF) (2009).

101. T / F When the disk arm moves from end to end, servicing all requests in its path, the scheduling mechanism is known as SCAN (2009).

102. T / F When the disk arm moves from end to end, servicing all requests in its path, the scheduling mechanism is known as LOOK (2009).

103. T / F PCI is a computer bus interface for attaching certain hardware devices (2009).

104. T / F USB is a computer bus interface for attaching certain hardware devices (2009).

105. T / F SATA is a computer bus interface for attaching certain hardware devices (2009).