

## 1. 2016-1

Circle or cross: "T" if True – "F" if False.

- T / F A semaphore is a data structure.
- T / F Semaphores can not be used for avoiding dead locks
- T / F A monitor is a programming language construct
- T / F Monitors encapsulate shared data structures.
- T / F Both semaphores and monitors are distributed as function calls.
- T / F Monitors use condition variables, while semaphores do not.

## 2. 2016-2

```
001 /*
002  * (c) 2015-2016 Rahmat M. Samik-Ibrahim
002  * -- This is free software
003  * Feel free to copy and/or modify and/
004  * or distribute it, provided this notice,
004  * and the copyright notice, are preserved.
005  * REV04 Tue Dec 13 15:19:04 WIB 2016
006  * START Wed Sep 30 00:00:00 UTC 2015
007  */
008
009 #include <stdio.h>
010 #include <stdlib.h>
011 #include <semaphore.h>
012 #include "99-myutils.h"
013 #define nSem 7
014
015 sem_t sem[nSem];
016
017 void* thread1 (void* a) {
018     sem_wait (&sem[1]);
019     printf("T1X\n");
020     sem_post (&sem[4]);
021 }
022
023 void* thread2 (void* a) {
024     sem_wait (&sem[2]);
025     printf("T2X\n");
026     sem_post (&sem[5]);
027     sem_post (&sem[1]);
028 }
030 void* thread3 (void* a) {
031     printf("T3X\n");
032     sem_post (&sem[6]);
033     sem_post (&sem[2]);
034 }
035
036 void* thread4 (void* a) {
037     sem_wait (&sem[4]);
038     printf("T44\n");
039     sem_wait (&sem[5]);
040     printf("T45\n");
041     sem_wait (&sem[6]);
042     printf("T46\n");
043 }
044
045 void main(void) {
046     printf("MAIN\n");
047     for (int ii=1;ii<nSem;ii++)
048         sem_init(&sem[ii], 0, 0);
049     daftar_trit (thread1);
050     daftar_trit (thread2);
051     daftar_trit (thread3);
052     daftar_trit (thread4);
053     jalankan_trit ();
054     beberes_trit ("TREXIT");
055 }
```

Write down the program output:

## 3. 2017-1

Program Code of Synchronization (using 99-myutils.h and 99-myutils.c from the lab assignment)	
<pre> 001 /* (c) 2011-2017 Rahmat M. Samik-Ibrahim 002  * This is free software. Feel free to copy and/or 003  * modify and/or distribute it, provided this 004  * notice, and the copyright notice, are preserved. 005  * REV01 Wed May 17 17:02:37 WIB 2017 006  * START Wed May 3 12:58:28 WIB 2017 007  * 008  * sem_init(), sem_wait(), sem_post(): semaphore 009  * sleep(X): sleep X seconds 010  * daftar_trit(T): register thread T 011  * jalankan_trit(): start all registered threads. 012  * beberes_trit(): exit all threads above.  */ 013 #define jmlKIRI    3 014 #define jmlKANAN   2 015 #define SLEEP      1 016 #include &lt;stdio.h&gt; 017 #include &lt;stdlib.h&gt; 018 #include &lt;semaphore.h&gt; 019 #include &lt;unistd.h&gt; 020 #include "99-myutils.h" 021 sem_t  mutexID, syncModKiri, syncModKanan; 022 sem_t  syncKiriMod, syncKananMod; 023 int    sequence = 0; 024 025 void cetak(char* posisi) { 026     sem_wait (&amp;mutexID); 027     printf("%s (%d)\n", posisi, sequence++); 028     fflush(NULL); 029     sem_post (&amp;mutexID); 030     sleep(SLEEP); 031 } 032 void* Kanan (void* a) { 033     while (TRUE) { 034         sem_wait (&amp;syncModKanan); 035         cetak("--++Kanan"); 036         sem_post (&amp;syncKananMod); 037     } 038 } </pre>	<pre> 039 void* Kiri (void* a) { 040     while (TRUE) { 041         cetak("Kiri-+-+--"); 042         sem_post (&amp;syncKiriMod); 043         sem_wait (&amp;syncModKiri); 044     } 045 } 046 void* Moderator (void* a) { 047     int ii; 048     while (TRUE) { 049         for (ii=0; ii&lt;jmlKIRI; ii++) 050             sem_wait (&amp;syncKiriMod); 051         for (ii=0; ii&lt;jmlKANAN; ii++) 052             sem_post (&amp;syncModKanan); 053         for (ii=0; ii&lt;jmlKANAN; ii++) 054             sem_wait (&amp;syncKananMod); 055         for (ii=0; ii&lt;jmlKIRI; ii++) 056             sem_post (&amp;syncModKiri); 057     } 058 } 059 int main(int argc, char * argv[]) { 060     int ii; 061     sem_init (&amp;syncModKiri, 0, 0); 062     sem_init (&amp;syncModKanan, 0, 0); 063     sem_init (&amp;syncKiriMod, 0, 0); 064     sem_init (&amp;syncKananMod, 0, 0); 065     sem_init (&amp;mutexID, 0, 1); 066 067     for (ii = 0 ; ii &lt; jmlKANAN; ii++) 068         daftar_trit(Kanan); 069     for (ii = 0 ; ii &lt; jmlKIRI; ii++) 070         daftar_trit(Kiri); 071     daftar_trit(Moderator); 072 073     jalankan_trit(); 074     beberes_trit("Selese..."); 075 } </pre>

Write down the next 5 lines of the program output:

K i r i - + - + - + - ( 0 )

---

---

---

---

---

---

---

---

---

---

#### 4. 2017-2

	column	column	column	column
row	1			
row				
row				
row			boxes	

**VALUE = 1**

```
cellSudoku[1][1][0] = 1;
cellSudoku[1][1][1] = 0;
cellSudoku[1][1][2] = 0;
cellSudoku[1][1][3] = 0;
cellSudoku[1][1][4] = 0;
```

**GUESSES = 2 3 4**

```
cellSudoku[1][2][0] = 0;
cellSudoku[1][2][1] = 0;
cellSudoku[1][2][2] = 2;
cellSudoku[1][2][3] = 3;
cellSudoku[1][2][4] = 4;
```

In this mini-Sudoku 4x4 — each **column**, **row**, and 2x2 sub-grid **box** — should contain the digits of: **1, 2, 3, or 4**. This C program "07-mini-sudoku-4x4.c" is using a 3 dimensional array called "cellSudoku[ ][ ][ ]". If "cellSudoku[row][column][0] == 0" (or: no value), "cellSudoku[row][column][1]" to "[4]" will contain of all values that are possible (or guesses).

- How many Semaphores were created in that program?
- Specify what the names of those Semaphores are!
- How many threads were created in that program?
- Specify what the (unique) names of those threads are!
- How many critical zone(s) are there in that program?
- Specify the line numbers of those critical zone(s)!
- Name the function that receives the input file "07-data.txt" in that program above!

**Program Code 07-mini-sudoku-4x4.c (using 99-myutils.h and 99-myutils.c from the DEMO set.)**

```

001 /*
002  * (c) 2017 Rahmat M. Samik-Ibrahim
003  * http://rahmatm.samik-ibrahim.vlsm.org/
004  * This is free software.
005  * REV04 Tue Dec 12 20:35:44 WIB 2017
006  * START Mon Dec 4 18:52:57 WIB 2017
007  */
008
009 #include <stdio.h>
010 #include <stdlib.h>
011 #include <unistd.h>
012 #include "99-myutils.h"
013 #define WaitSudoku 3
014 #define SSIZE 4
015 #define TOTALSIZE SSIZE * SSIZE
016
017 int globalExit=FALSE;
018 sem_t mutexing;
019 sem_t syncing1;
020 sem_t syncing2;
021
022 // cellSudoku[row][column][0] = value
023 // cellSudoku[row][column][1-4] = guesses
024 // if (value != 0) all guesses = 0
025 // (no more guesses)
026 int cellSudoku[][SSIZE+1][SSIZE+1]={
027     {},{ {}, {0,1,2,3,4}, {0,1,2,3,4},
028           {0,1,2,3,4}, {0,1,2,3,4}},
029     { {}, {0,1,2,3,4}, {0,1,2,3,4},
030           {0,1,2,3,4}, {0,1,2,3,4}},
031     { {}, {0,1,2,3,4}, {0,1,2,3,4},
032           {0,1,2,3,4}, {0,1,2,3,4}},
033     { {}, {0,1,2,3,4}, {0,1,2,3,4},
034           {0,1,2,3,4}, {0,1,2,3,4}}
035 };
036
037 // Print Cells
038 void printCells(char* state) {
039     printf ("\nSudoku Cells: %s\n", state);
040     for ( int jj=1; jj<SSIZE+1; jj++) {
041         for (int kk=1; kk<SSIZE+1; kk++) {
042             int cell=cellSudoku[jj][kk][0];
043             if (cell == 0 || cell == 5)
044                 printf ("[ ]");
045             else printf ("[%d]", cell);
046             if (kk == SSIZE) printf ("\n");
047         }
048     }
049     fflush(NULL);
050 }
051
052 // Filling the CELLS
053 void
054 fillCell(int rowCell,int colCell,int valCell)
055 {
056     sem_wait (&mutexing);
057     // Filling "valCell" into
058     // cellSudoku[rowCell, colCell];
059     cellSudoku[rowCell][colCell][0] = valCell;
060     // This is Cell is "taken".
061     // Eliminate all guesses!
062     for (int ii=1; ii<SSIZE+1; ii++) {
063         cellSudoku[rowCell][colCell][ii] = 0;
064     }
065     // Deleting "valCell"
066     // from all "columns guess"
067     for (int ii=1; ii<SSIZE+1; ii++) {
068         cellSudoku[rowCell][ii][valCell] = 0;
069     }
070     // Delete "valCell" from all "rows guess".
071     for (int ii=1; ii<SSIZE+1; ii++) {
072         cellSudoku[ii][colCell][valCell] = 0;
073     }
074     // Delete "valCell" from all "boxes guess".
075     rowCell = 1 + 2*((rowCell - 1)/2);
076     colCell = 1 + 2*((colCell - 1)/2);
077     for (int ii=rowCell; ii<rowCell+2; ii++) {
078         for (int jj=colCell; jj<colCell+2; jj++){
079             cellSudoku[ii][jj][valCell] = 0;
080         }
081     }
082     sem_post (&mutexing);
083 }
084
085 // From Standard Input into Cell using
086 // fillCell -- SCAN INPUT: scanf()
087 // is the oposite of printf()
088 void inputCell(void) {
089     for (int ii=0; ii < TOTALSIZE; ii++) {
090         int tmpCell=0;
091         scanf("%d", &tmpCell);
092         int rowCell = ii/4 + 1;
093         int colCell = ii%4 + 1;
094         if (tmpCell != 0) {
095             fillCell(rowCell,colCell,tmpCell);
096         }
097     }
098 }

```

Program Code 07-mini-sudoku-4x4.c (using 99-myutils.h and 99-myutils.c from the DEMO set.)	
<pre> 100 // CellWatcher 101 int cwID = 0; 102 void* cellWatcher (void* a) { 103     sem_wait (&amp;syncing1); 104     sem_wait (&amp;mutexing); 105     int rowCell = cwID/4 + 1; 106     int colCell = cwID%4 + 1; 107     cwID++; 108     sem_post (&amp;mutexing); 109     int localExit=FALSE; 110     while (!localExit &amp;&amp; !globalExit) { 111         int tmpCell=0, nZero=0; 112         for (int ii=1; ii&lt;SSIZE+1; ii++) { 113             if(cellSudoku[rowCell][colCell][ii]==0) 114                 nZero++; 115             else 116                 tmpCell=ii; 117         } 118         if (nZero==3) 119             fillCell(rowCell, colCell, tmpCell); 120         localExit = 121             cellSudoku[rowCell][colCell][0]!=0; 122     } 123     fflush(NULL); 124     sem_post (&amp;syncing2); 125 } 126 127 // Timeout after "WaitSudoku" 128 void* managerSudoku (void* a) { 129     sleep(WaitSudoku); 130     for (int ii=0; ii&lt;TOTALSIZE; ii++) { 131         int rowCell = ii/4 + 1; 132         int colCell = ii%4 + 1; 133         if(cellSudoku[rowCell][colCell][0]==0){ 134             cellSudoku[rowCell][colCell][0] = 5; 135         } 136         sem_post (&amp;syncing2); 137     } 138     globalExit = TRUE; 139 } </pre>	<pre> 141 // Display Sudoku 142 void* displaySudoku (void* a) { 143     printCells("INITIAL"); 144     for(int jj=0;jj&lt;TOTALSIZE;jj++) 145         sem_post(&amp;syncing1); 146     for(int jj=0;jj&lt;TOTALSIZE;jj++) 147         sem_wait(&amp;syncing2); 148     printCells("RESULT"); 149 } 150 151 // This is MAIN 152 void main(void) { 153     printf ("MAIN: START\n"); 154     sem_init (&amp;mutexing, 0, 1); 155     sem_init (&amp;syncing1, 0, 0); 156     sem_init (&amp;syncing2, 0, 0); 157     inputCell(); 158     for (int ii=0; ii&lt;TOTALSIZE; ii++) { 159         daftar_trit(cellWatcher); 160     } 161     daftar_trit (displaySudoku); 162     daftar_trit (managerSudoku); 163     jalankan_trit (); 164     beberes_trit ("\\nTRIT: EXIT"); 165 } </pre>
<p>This following is the output of executing: ./07-mini-sudoku-4x4 &lt; 07-data.txt</p>	<p><b>Bonus Question:</b> What is inside file 07-data.txt ?</p>
<pre> MAIN: START  Sudoku Cells: INITIAL [ ][ ][ ][3] [ ][1][4][ ] [ ][2][3][ ] [1][ ][ ][ ]  Sudoku Cells: RESULT [2][4][1][3] [3][1][4][2] [4][2][3][1] [1][3][2][4]  TRIT: EXIT </pre>	

## 5. 2018-1

```

01  /*
02  Copyright 2018 Rahmat M. Samik-Ibrahim
03  You are free to SHARE (copy and
04  redistribute the material in any medium
05  or format) and to ADAPT (remix,
06  transform, and build upon the material
07  for any purpose, even commercially).
08  This program is distributed in the hope
09  that it will be useful, but WITHOUT ANY
10  WARRANTY; without even the implied
11  warranty of MERCHANTABILITY or FITNESS
12  FOR A PARTICULAR PURPOSE.
13
14  * REV02 Wed May  2 11:30:19 WIB 2018
15  * START Wed Apr 18 19:50:01 WIB 2018
16  */
17
18  // DO NOT USE THE SAME SEMAPHORE NAME!!!!
19  // Replace "demo" with your own SSO name.
20  #define SEM_COUNT1      "/count-1-demo"
21  #define SEM_COUNT2      "/count-2-demo"
22  #define SEM_MUTEX       "/mutex-demo"
23  #define SEM_SYNC        "/sync-demo"
24
25  #include <fcntl.h>
26  #include <stdio.h>
27  #include <stdlib.h>
28  #include <unistd.h>
29  #include <semaphore.h>
30  #include <sys/mman.h>
31  #include <sys/types.h>
32  #include <sys/wait.h>
33
34  // Shared Memory: R/W with no name.
35  #define PROT      (PROT_READ | PROT_WRITE)
36  #define VISIBLE   (MAP_ANONYMOUS|MAP_SHARED)
37
38  #define LOOP      2
39  #define BUFSIZE   1
40
41  sem_t*   ctr_prod;
42  sem_t*   ctr_cons;
43  sem_t*   mutex;
44  sem_t*   ssync;
45  int*     product;
46
47  // WARNING: NO ERROR CHECK! //////////////////
48  void flushprintf(char* str, int ii) {
49      printf("%s [%d]\n", str, ii);
50      fflush(NULL);
51  }
52
53  void init(void) {
54      product = mmap(NULL, sizeof(int),
55                      PROT, VISIBLE, 0, 0);
56      *product = 0;
57      ctr_prod = sem_open(SEM_COUNT1,
58                          O_CREAT, 0600, BUFSIZE);
59      ctr_cons = sem_open(SEM_COUNT2,
60                          O_CREAT, 0600, 0);
61      mutex = sem_open(SEM_MUTEX,
62                       O_CREAT, 0600, 1);
63      ssync = sem_open(SEM_SYNC,
64                       O_CREAT, 0600, 0);
65  }
66
67  void producer (void) {
68      sem_wait(ssync);
69      flushprintf("PRODUCER PID",getpid());
70      for (int loop=0; loop<LOOP; loop++) {
71          sem_wait(ctr_prod);
72          sem_wait(mutex);
73          flushprintf("PRODUCT  ",
74                      ++(*product));
75
76          sem_post(mutex);
77          sem_post(ctr_cons);
78      }
79      wait(NULL);
80  }
81
82  void consumer (void) {
83      flushprintf("CONSUMER PID",getpid());
84      sem_post(ssync);
85      for (int loop=0; loop<LOOP; loop++) {
86          sem_wait(ctr_cons);
87          sem_wait(mutex);
88          flushprintf("CONSUME  ", *product);
89          sem_post(mutex);
90          sem_post(ctr_prod);
91      }
92  }
93
94  // WARNING: NO ERROR CHECK! //////////////////
95  void main(void) {
96      flushprintf("STARTING PID",getpid());
97      init();
98      if (fork()) producer(); // Parent
99      else        consumer(); // Child
100      sem_unlink(SEM_COUNT1);
101      sem_unlink(SEM_COUNT2);
102      sem_unlink(SEM_SYNC);
103      sem_unlink(SEM_MUTEX);
104      flushprintf("STOP HERE PID", getpid());
105  }

```

**6. 2018-1 (continued)...**

- (a) Assume the Parent PID is 1000 and the Child PID is 1001. What is the output of the program above?
- (b) Name all four (4) semaphore!
- (c) What is the purpose of line 68?
- (d) What is the purpose of line 71?

- (e) What is the purpose of line 77?
- (f) What is the purpose of line 84?
- (g) How many Critical Section(s) is/are there in the program above? Where/which lines are the Critical Section(s)?
- (h) Explain briefly the purpose of function `fflush(NULL)` in line 50!
- (i) What is the purpose of lines 98 - 101?

**7. 2018-2** (See <https://rms46.vlsm.org/2/201.pdf> 2018-2)

- (a) Name all three (3) semaphores!
- (b) What is the purpose of lines 65 & 66?
- (c) What is the purpose of lines 74 & 75?

- (d) What is the purpose of lines 88, 89, 113 & 114?
- (e) What is the purpose of line 90 in regard of lines 91 & 115
- (f) Is there any Critical Section(s) in the program (Yes/No)? If "Yes", which line(s)?

**8. 2019-1** (See <https://rms46.vlsm.org/2/201.pdf> 2019-1)

- (a) What is the purpose of semaphores "turns[ ]" (See lines 79-80, 94, 95)?
- (b) What is the purpose of semaphore "mutex" (See lines 82, 100, 106)?
- (c) Explain why the final "**rTime**" value of each rider will always be unique (See lines 101-104)!
- (d) Explain why the program output will always be printed in a proper "**rTime**" order (See lines 101-104)!

**9. 2019-2** (See <https://rms46.vlsm.org/2/201.pdf> 2019-2)

Modify lines 66 to 74, so that the child of `fork()` (line 66), will be executed first at line 75.

-----