

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 /*                                030 void* thread3 (void* a) {
002 * (c) 2015-2016 Rahmat M. Samik-Ibrahim 031     printf("T3X\n");
002 * -- This is free software             032     sem_post (&sem[6]);
003 * Feel free to copy and/or modify and/ 033     sem_post (&sem[2]);
004 * or distribute it, provided this notice, 034 }
004 * and the copyright notice, are preserved. 035
005 * REV04 Tue Dec 13 15:19:04 WIB 2016    036 void* thread4 (void* a) {
006 * START Wed Sep 30 00:00:00 UTC 2015    037     sem_wait (&sem[4]);
007 */                                       038     printf("T44\n");
008                                         039     sem_wait (&sem[5]);
009 #include <stdio.h>                       040     printf("T45\n");
010 #include <stdlib.h>                       041     sem_wait (&sem[6]);
011 #include <semaphore.h>                   042     printf("T46\n");
012 #include "99-myutils.h"                 043 }
013 #define nSem 7                           044
014                                         045 void main(void) {
015 sem_t sem[nSem];                         046     printf("MAIN\n");
016                                         047     for (int ii=1;ii<nSem;ii++)
017 void* thread1 (void* a) {                048         sem_init(&sem[ii], 0, 0);
018     sem_wait (&sem[1]);                   049     daftar_trit (thread1);
019     printf("T1X\n");                       050     daftar_trit (thread2);
020     sem_post (&sem[4]);                   051     daftar_trit (thread3);
021 }                                         052     daftar_trit (thread4);
022                                         053     jalankan_trit ();
023 void* thread2 (void* a) {                054     beberes_trit ("T3EXIT");
024     sem_wait (&sem[2]);                   055 }
025     printf("T2X\n");
026     sem_post (&sem[5]);
027     sem_post (&sem[1]);
028 }
```

Write down the program output:

4. 2017-2

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  * REVO4 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.)

```

100 // CellWatcher
101 int cwID = 0;
102 void* cellWatcher (void* a) {
103     sem_wait (&syncing1);
104     sem_wait (&mutexing);
105     int rowCell = cwID/4 + 1;
106     int colCell = cwID%4 + 1;
107     cwID++;
108     sem_post (&mutexing);
109     int localExit=FALSE;
110     while (!localExit && !globalExit) {
111         int tmpCell=0, nZero=0;
112         for (int ii=1; ii<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 (&syncing2);
125 }
126
127 // Timeout after "WaitSudoku"
128 void* managerSudoku (void* a) {
129     sleep(WaitSudoku);
130     for (int ii=0; ii<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 (&syncing2);
137     }
138     globalExit = TRUE;
139 }
141 // Display Sudoku
142 void* displaySudoku (void* a) {
143     printCells("INITIAL");
144     for(int jj=0;jj<TOTALSIZE;jj++)
145         sem_post(&syncing1);
146     for(int jj=0;jj<TOTALSIZE;jj++)
147         sem_wait(&syncing2);
148     printCells("RESULT");
149 }
150
151 // This is MAIN
152 void main(void) {
153     printf ("MAIN: START\n");
154     sem_init (&mutexing, 0, 1);
155     sem_init (&syncing1, 0, 0);
156     sem_init (&syncing2, 0, 0);
157     inputCell();
158     for (int ii=0; ii<TOTALSIZE; ii++) {
159         daftar_trit(cellWatcher);
160     }
161     daftar_trit (displaySudoku);
162     daftar_trit (managerSudoku);
163     jalankan_trit ();
164     beberes_trit ("\nTRIT: EXIT");
165 }

```

This following is the output of executing:
 ./07-mini-sudoku-4x4 < 07-data.txt

Bonus Question:
 What is inside file 07-data.txt ?

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