

CSGE602055 Operating Systems

CSF2600505 Sistem Operasi

Minggu 07: Synchronization

Rahmat M. Samik-Ibrahim

Universitas Indonesia

<http://rms46.vlsm.org/2/207.html>

REV62 2-Sep-2017

Minggu 00	29 Aug - 05 Sep 2017	Intro & Review
Minggu 01	07 Sep - 12 Sep 2017	IPR, REGEX, & Scripting
Minggu 02	14 Sep - 19 Sep 2017	Protection, Security, Privacy, & C-language
Minggu 03	26 Sep - 30 Sep 2017	BIOS, Loader, Systemd, & I/O
Minggu 04	03 Okt - 07 Okt 2017	Addressing, Shared Lib, Pointer & I/O Programming
Minggu 05	10 Okt - 14 Okt 2017	Virtual Memory
Ming. UTS	15 Okt - 24 Okt 2017	
Minggu 06	26 Okt - 31 Okt 2017	Concurrency: Processes & Threads
Minggu 07	02 Nov - 07 Nov 2017	Synchronization
Minggu 08	09 Nov - 14 Nov 2017	Scheduling & Network Sockets Programming
Minggu 09	16 Nov - 21 Nov 2017	File System & Persistent Storage
Minggu 10	23 Nov - 28 Nov 2017	Special Topic: Blockchain
Cadangan	30 Nov - 09 Des 2017	
Ming. UAS	10 Des - 23 Des 2017	

Agenda

- 1 Start
- 2 Agenda
- 3 Week 07
- 4 Peterson
- 5 myutils.h
- 6 myutils.c
- 7 Bounded Buffer
- 8 Readers Writers
- 9 Rock Paper Scissors Lizard Spock
- 10 Lab
- 11 The End

Week 07: Synchronization

- Reference: (OSCE2e ch5) (UCB 7/8) (UDA P3L3/4) (OLD 04)
- The Critical Section Problem
- Race Condition
- Peterson's Solution
- Semaphores
- Deadlock and Starvation
- Deadlock Characterization
- Resource Graph

Peterson's Solution

Process 0

flag[0]=

turn=

```
do {  
    flag[0] = true  
    turn = 1  
    while (flag[1] && turn == 1)  
        (do nothing);  
    [CRITICAL SECTION];  
    flag[0] = false  
    [REMAINDER SECTION];  
} while(true);
```

Process 1

flag[1]=

```
do {  
    flag[1] = true  
    turn = 0  
    while (flag[0] && turn == 0)  
        (do nothing);  
    [CRITICAL SECTION];  
    flag[1] = false  
    [REMAINDER SECTION];  
} while(true);
```

```
/*
 * (c) 2011-2016 Rahmat M. Samik-Ibrahim
 * -- This is free software
 * Feel free to copy and/or modify and/or
 * distribute it, provided this notice, and
 * the copyright notice, are preserved.
 * REV02 Mon Nov 7 14:33:08 WIB 2016
 * REV01 Wed Nov 2 11:50:14 WIB 2016
 * REV00 Xxx Sep 30 XX:XX:XX UTC 2015
 * START Xxx Mar 30 02:13:01 UTC 2011
 */

#define MAX_THREAD 256
#define BUFFER_SIZE 5
#define TRUE 1
#define FALSE 0
```

```
typedef struct {
    int    buffer[BUFFER_SIZE];
    int    in;
    int    out;
    int    count;
} bbuf_t;

// mempersiapkan "trit"
void daftar_trit    (void* trit);

// menjalankan dan menunggu hasil dari "daftar_trit"
void jalankan_trit (void);

// beberes menutup "jalankan_trit"
void beberes_trit  (char* pesan);
```

```
// istirohat acak "0-max_mdetik" (ms)
void rehat_acak      (long max_mdetik);

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
```



```
/*
 * (c) 2011-2016 Rahmat M. Samik-Ibrahim
 * -- This is free software
 * Feel free to copy and/or modify and/or
 * distribute it, provided this notice, and
 * the copyright notice, are preserved.
 * REV01 Wed Nov  2 11:49:55 WIB 2016
 * REV00 Xxx Sep 30 XX:XX:XX UTC 2015
 * START Xxx Mar 30 02:13:01 UTC 2011
 */
#include <pthread.h>
#include <semaphore.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include "99-myutils.h"
```

daftar_trit()

```
sem_t      mutex, db, empty, full, rmutex, wmutex;
int        jumlah_trit = 0;
void*      trits  [MAX_THREAD];
pthread_t  trit_id[MAX_THREAD];
void daftar_trit(void *trit) {
    if(jumlah_trit >= MAX_THREAD) {
        printf("\n ERROR MAX daftar_trit %d\n",jumlah_trit);
        exit(1);
    }
    trits[jumlah_trit++] = trit;
}
void beberes_trit(char* pesan) {
    if (pesan != NULL)
        printf("%s\n",pesan);
    pthread_exit(NULL);
}
```

jalankan_trit()

```
void jalankan_trit(void){
    int ii;
    for (ii=0;ii<jumlah_trit;ii++) {
        if(pthread_create(&trit_id[ii], NULL, trits[ii], NULL))
            printf("\n ERROR pthread_creat: %d\n",ii);
            exit(1);
        }
    }
    for (ii=0;ii<jumlah_trit;ii++){
        if(pthread_join(trit_id[ii], NULL)) {
            printf("\n ERROR pthread_join: %d\n",ii);
            exit(1);
        }
    }
}
```

rehat_acak()

```
/* REHAT *****/
int  pertamax    = TRUE;

void rehat_acak(long max_mdetik) {
    struct timespec tim;
    long          ndetik;

    if (pertamax) {
        pertamax = FALSE;
        srandom((unsigned int) time (NULL));
    }
    ndetik      = random() % max_mdetik;
    tim.tv_sec   = ndetik   / 1000L;
    tim.tv_nsec  = ndetik   % 1000L * 1000000L;
    nanosleep(&tim, NULL);
}
```

init_buffer()

```
/* BOUNDED BUFFER *****/
bbuf_t buf;

void init_buffer(void) {
    buf.in    = 0;
    buf.out   = 0;
    buf.count = 0;
    sem_init  (&mutex, 0, 1);
    sem_init  (&empty, 0, BUFFER_SIZE);
    sem_init  (&full, 0, 0);
}
```

enter_buffer()

```
void enter_buffer(int entry) {  
    sem_wait(&empty);  
    sem_wait(&mutex);  
    buf.count++;  
    buf.buffer[buf.in] = entry;  
    buf.in = (buf.in+1) % BUFFER_SIZE;  
    sem_post(&mutex);  
    sem_post(&full);  
}
```

remove_buffer()

```
int remove_buffer(void) {  
    int item;  
    sem_wait(&full);  
    sem_wait(&mutex);  
    buf.count--;  
    item = buf.buffer[buf.out];  
    buf.out = (buf.out+1) % BUFFER_SIZE;  
    sem_post(&mutex);  
    sem_post(&empty);  
    return item;  
}
```

init_rw()

```
/* READERS WRITERS *****/

int readerCount;

void init_rw(void) {
    readerCount = 0;
    sem_init    (&mutex, 0, 1);
    sem_init    (&rmutex, 0, 1);
    sem_init    (&wmutex, 0, 1);
    sem_init    (&db,      0, 1);
}
```


startRead() — endRead()

```
int startRead(void) {  
    sem_wait(&mutex);  
    if (++readerCount == 1 )  
        sem_wait(&db);  
    sem_post(&mutex);  
    return readerCount;  
}
```

```
int endRead(void) {  
    sem_wait(&mutex);  
    if (--readerCount == 0 )  
        sem_post(&db);  
    sem_post(&mutex);  
    return readerCount;  
}
```

startWrite() — endWrite()

```
void startWrite(void) {  
    sem_wait(&db);  
}
```

```
void endWrite(void) {  
    sem_post(&db);  
}
```

Rock Paper Scissors Lizard Spock

```
/*  
 * (c) 2014-2016 Rahmat M. Samik-Ibrahim  
 * -- This is free software  
 * Feel free to copy and/or modify and/or  
 * distribute it, provided this notice, and  
 * the copyright notice, are preserved.  
 * REV01 Wed Nov  2 11:20:30 WIB 2016  
 * REV00 Xxx Sep 30 XX:XX:XX UTC 2015  
 * START Xxx Oct 19 XX:XX:XX UTC 2014  
 */
```

RPSLS 1

```
// *Rock*Paper*Scissors*Lizard*Spock*  
// Invented by Sam Kass and Karen Bryla  
// Rock crushes Scissors  
// Rock crushes Lizard  
// Paper covers Rock  
// Paper disproves Spock  
// Scissors cut Paper  
// Scissors decapitate Lizard  
// Lizard eats Paper  
// Lizard poisons Spock  
// Spock vaporizes Rock  
// Spock smashes Scissors
```

```
#include <semaphore.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <unistd.h>
#include "99-myutils.h"
#define nPlayers 2
#define nWeapons 5
int      playerSEQ=1;
int      myWeapon[nPlayers+1];
sem_t    mutex, sync1, sync2;

// (0=Rock) (1=Paper) (2=Scissors) (3=Lizard) (4=Spock)
char* weaponName[nWeapons]= {
    "Rock", "Paper", "Scissors", "Lizard", "Spock"
};
```

```
// '-' = draw  'v' = win  'x' = lose
char weaponTable[nWeapons][nWeapons] = {
    {'-', 'x', 'v', 'v', 'x'},
    {'v', '-', 'x', 'x', 'v'},
    {'x', 'v', '-', 'v', 'x'},
    {'x', 'v', 'x', '-', 'v'},
    {'v', 'x', 'v', 'x', '-'}
};

void waitPlayers() {
    for (int ii=0; ii < nPlayers; ii++)
        sem_wait(&sync1);
}

void postPlayers() {
    for (int ii=0; ii < nPlayers; ii++)
        sem_post(&sync2);
}
```

```
void* playerThread (void* a) {
    int      playerID;
    sem_wait (&mutex);
    playerID=playerSEQ++;
    sem_post (&mutex);
    printf("Player[%d]: READY\n",playerID);
    sem_post (&sync1);
    sem_wait (&sync2);
    myWeapon[playerID] = rand() % nWeapons;
    printf("Player[%d]: %s\n",
        playerID, weaponName[myWeapon[playerID]]);
    sem_post (&sync1);
}
```

```
void* refereeThread (void* a) {
    waitPlayers();
    printf("Referee:    ALL READY!\n");
    postPlayers();
    waitPlayers();
    char result =
        weaponTable[myWeapon[1]][myWeapon[2]];
    if (result == '-')
        printf("Referee:    DRAW!\n");
    else if (result == 'v')
        printf("Referee:    Player[1] WINS!\n");
    else
        printf("Referee:    Player[2] WINS!\n");
}
```



```
void main() {  
    // randomize with a time seed  
    srand(time(NULL));  
    sleep(1);  
    // init semaphore mutex = 1 syncx = 0  
    sem_init (&mutex, 0, 1);  
    sem_init (&sync1, 0, 0);  
    sem_init (&sync2, 0, 0);  
    // register and execute threads  
    daftar_trit (refereeThread);  
    for (int ii=0; ii<nPlayers; ii++)  
        daftar_trit (playerThread);  
    jalankan_trit ();  
    beberes_trit ("Goodbye...");  
}
```

- semaphores()
- pthread()

The End

- This is the end of the presentation.