

# Sincronizzazione / Synchronization

## Exam 2020/06/16 - Ex 1 (6.0 points)

### Italiano

Si indichi l'output o gli output possibili che possono essere ottenuti eseguendo concorrentemente i processi P , P e P riportato di seguito.

### English

Indicate the possible output or outputs that can be obtained by concurrently executing the processes P , P e P as follow

```
init (S1, 1); init(S2, 0); init(S3, 0);
```

#### PA

```
wait (S2);  
printf("F");  
signal (S1);  
wait (S2);  
printf("G");
```

#### PB

```
wait (S1);  
printf("B");  
signal (S3);  
wait (S1);  
printf("D");  
signal (S2);
```

#### PC

```
printf("A");  
wait (S3);  
printf("C");  
signal (S1);  
wait (S1);  
printf("E");  
signal (S2);
```

Scegli una o più alternative: [Choose one or more options:](#)

1. ☒ B A C E F D G
2. ☐ A C B D F E G
3. ☒ B A C D F E G
4. ☒ A B C D F E G
5. ☐ A C B E D F G
6. ☒ A B C E F D G
7. ☐ B A C E D F G
8. ☐ A C B E F D G

## Exam 2020/06/16 - Ex 3 (3.0 points)

### Italiano

Si indichino quali sono le caratteristiche dell'implementazione del paradigma produttore/consumatore riportato alla fine della domanda, ed eseguito concorrentemente da più produttori e consumatori. Segnare tutte le affermazioni vere

### English

Indicate which are the characteristics of the implementation of the producer/consumer paradigm reported at the end of the question, and executed concurrently by many producers and consumers. Mark all the true answers.

```
init (full, 0); init (empty, SIZE); init (mutex, 1);
```

```
Producer () {  
    int val;  
    while (TRUE) {  
        produce (&val);  
        wait (empty);  
        wait (mutex);  
        enqueue (val);  
        signal (mutex);  
        signal (full);  
    }  
}
```

```

}

Consumer () {
    int val;
    while (TRUE) {
        wait (mutex);
        wait (full);
        dequeue (&val);
        signal (mutex);
        signal (empty);
        consume (val);
    }
}

```

Scegli una o più alternative: [Choose one or more options:](#)

1. ☒ E' soggetta a starvation. [It is subject to starvation.](#)
2. ☒ Limita la concorrenza dei processi. [It limits processes concurrency.](#)
3. ☒ E' soggetta a deadlock. [It is subject to deadlock.](#)

### Exam 2021/09/07 - Ex 8 (3.0 points)

#### Italiano

Si analizzi il seguente tratto di codice. Si indichi qual è l'output generato dal programma. Si osservi che risposte errate implicano una penalità nel punteggio finale.

#### English

[Analyze the following segment of code. Indicate which is the output generated by the program. Note that incorrect answers imply a penalty in the final score.](#)

```

#include <stdio.h>
#include <stdlib.h>
#include <semaphore.h>
#include <pthread.h>

sem_t *sa, *sb, *sc, *me, *ss;
int n;

static void *TA ();
static void *TB ();
static void *TC ();

int main (int argc, char **argv) {
    pthread_t th;
    sa = (sem_t *) malloc (sizeof(sem_t));
    sb = (sem_t *) malloc (sizeof(sem_t));
    sc = (sem_t *) malloc (sizeof(sem_t));
    me = (sem_t *) malloc (sizeof(sem_t));
    ss = (sem_t *) malloc (sizeof(sem_t));
    n = 0;
    sem_init (sa, 0, 0);
    sem_init (sb, 0, 0);
    sem_init (sc, 0, 0);
    sem_init (me, 0, 1);
    sem_init (ss, 0, 2);
    setbuf(stdout, 0);
    pthread_create (&th, NULL, TA, NULL);
    pthread_create (&th, NULL, TB, NULL);

```

```

        pthread_create (&th, NULL, TC, NULL);
        pthread_exit(0);
    }

static void *TA () {
    pthread_detach (pthread_self ());
    while (1) {
        sem_wait (ss);
        sem_wait (me);
        printf ( "A");
        sem_post (me);
        n++;
        if (n==2)
            sem_post (sc);
        sem_wait (sa);
    } return 0;
}

static void *TB () {
    pthread_detach (pthread_self ());
    while (1) {
        sem_wait (ss);
        sem_wait (me);
        printf ( "B");
        sem_post (me);
        n++;
        if (n==2)
            sem_post (sc);
        sem_wait (sb);
    }
    return 0;
}

static void *TC () {
    pthread_detach (pthread_self ());
    while (1) {
        sem_wait (sc);
        n = 0;
        printf ("C\n");
        sem_post (sa);
        sem_post (sb);
        sem_post (ss);
        sem_post (ss);
    }
    return 0;
}

```

Scegli UNA SOLA alternativa: [Choose JUST ONE option:](#)

1. ☐ Una sequenza di stringhe "ACB". [A sequence of strings "ACB".](#)
2. ☒ Una sequenza di stringhe "ABC" e "BAC". [A sequence of strings "ABC" and "BAC".](#)
3. ☐ Una sequenza di stringhe "ABC". [A sequence of strings "ABC".](#)
4. ☐ Una sequenza di stringhe "BCA". [A sequence of strings "BCA".](#)
5. ☐ Una sequenza di stringhe "BAC". [A sequence of strings "BAC".](#)

## Exam 2022/01/28 - Ex 10 (3.0 points)

### Italiano

Dato il seguente codice, si indichi quali delle seguenti affermazioni sono corrette. Si osservi che risposte errate implicano una penalità nel punteggio finale.

### English

Given the following code, indicate which of the following statements are correct. Note that incorrect answers imply a penalty in the final score.

```
typedef struct lock_s {
    int ticketNumber;
    int turnNumber;
} lock_t;
int increment (int *var) {
    int tmp = *var;
    *var = tmp + 1;
    return (tmp);
}
void init (lock_t lock) {
    lock.ticketNumber = 0;
    lock.turnNumber = 0;
}
void lock (lock_t lock) {
    int myTurn = increment (&lock.ticketNumber);
    while (lock.turnNumber != myTurn);
}
void unlock (lock_t lock) {
    increment (&lock.turnNumber);
}
```

Scegli una o più alternative: [Choose one or more options:](#)

- ☐ La funzione increment() chiamata da più processi concorrentemente soffre di race condition sulla variabile tmp. [The function increment\(\) called concurrently by many processes suffers of race conditions on the variable tmp.](#)
- ☐ Un processo o thread bloccato sulla funzione lock() è in stato di attesa e non usa la CPU. [A process or a thread blocked on the function lock\(\) is in a waiting state and does not use the CPU.](#)
- ☒ A seguito dell'esecuzione da parte di due processi della funzione lock(), la variabile lock.ticketNumber potrebbe essere aumentata di una sola unità (cioè se il suo valore iniziale è 0, quello finale potrebbe essere 1). [After the execution by means of two threads of the function lock\(\), the variable lock.ticketNumber could be incremented of only one unit \(i.e., if its initial value is 0, the final value could be 1\).](#)
- ☒ Le funzioni lock() e unlock() possono essere utilizzate per gestire l'accesso a una sezione critica nel caso in cui increment() sia eseguita in modo atomico. [Functions lock\(\) and unlock\(\) can be used to manage the access to a critical section in the case the function increment\(\) is executed in an atomic way.](#)

## Exam 2022/06/14 - Ex 7 (5.0 points)

### Italiano

Un numero imprecisato di thread può invocare una delle seguenti due funzioni:

- void wait\_ch(int x), con  $0 \leq x < 10$ . Questa funzione blocca il thread chiamante finché non viene invocata la funzione signal\_ch(int y), con  $y == x$ .
- void signal\_ch(int y), con  $0 \leq y < 10$ . Questa funzione risveglia tutti i thread che hanno invocato in precedenza la funzione wait\_ch(int x), con  $x == y$ . La funzione non ha effetto se nessun thread soddisfa la condizione enunciata al momento in cui essa viene chiamata.

Esempio di funzionamento:

- Il thread A chiama `wait_ch(4)` e viene posto in attesa.
- Il thread B chiama `wait_ch(2)` e viene posto in attesa.
- Il thread C chiama `signal_ch(8)` senza alcun effetto.
- Il thread D chiama `wait_ch(4)` e viene posto in attesa.
- Il thread E chiama `signal_ch(4)` risvegliando i thread A e D.
- Il thread D chiama `signal_ch(2)` risvegliando il thread B.

Si realizzino le funzioni `wait_ch()` e `signal_ch()` usando i semafori POSIX, definendone le relative variabili condivise.

#### English

An unknown number of threads can invoke one of the following two functions:

1. `void wait_ch(int x)`, with  $0 \leq x < 10$ . This function blocks the calling thread until the function `signal_ch(int y)` is called with  $y == x$ .
2. `void signal_ch(int y)`, with  $0 \leq y < 10$ . This function wakes up all the threads that previously called the function `wait_ch(int x)` with  $x == y$ . The function has no effect if no thread satisfies the condition previously described at the time it is called.

Example of operation:

- Thread A calls `wait_ch(4)` and it is blocked.
- Thread B calls `wait_ch(2)` and it is blocked.
- Thread C calls `signal_ch(8)` without any effect.
- Thread D calls `wait_ch(4)` and it is blocked.
- Thread E calls `signal_ch(4)` that wakes up threads A and D.
- Thread D calls `signal_ch(2)` that wakes up thread B.

Implement the functions `wait_ch()` and `signal_ch()` using POSIX semaphores, defining the related shared variables.

Risposta: Answer:

```
// Solution 1 (CORRECT WITH THE ALLOWED HYPOTHESIS OF ACYCLIC THREADS)
#define MAX_WQ 10

int n_wait[MAX_WQ];
sem_t sync[MAX_WQ];
sem_t m[MAX_WQ];

int main() {
    for (int i=0; i<MAX_WQ; i++){
        n_wait[i] = 0;
        sem_init(&sync[i], 0, 0);
        sem_init(&m[i], 0, 1);
    }

    ...
}

void wait_ch(int x) {
    sem_wait(&m[x]);
    n_wait[x]++;
    sem_post(&m[x]);

    sem_wait(&sync[x]);
}
```

```

}

void signal_ch(int y){
    sem_wait(&m[y]);
    for (int i=0; i<n_wait[y]; i++){
        sem_post(&sync[y]);
    }
    n_wait[y]=0;
    sem_post(&m[y]);
}

// Solution 2 (CORRECT WITH THE ALLOWED HYPOTHESIS OF CYCLIC THREADS)
#define MAX_WQ 10

int n_wait[MAX_WQ];
sem_t sync[MAX_WQ];
sem_t m[MAX_WQ];
sem_t b[MAX_WQ];

int main(){
    for (int i=0; i<MAX_WQ; i++){
        n_wait[i] = 0;
        sem_init(&sync[i], 0, 0);
        sem_init(&m[i], 0, 1);
        sem_init(&b[i], 0, 1);
    }

    ...
}

void wait_ch(int x){
    sem_wait(&b[x]);
    sem_wait(&m[x]);
    n_wait[x]++;
    sem_post(&m[x]);
    sem_post(&b[x]);

    sem_wait(&sync[x]);

    sem_wait(&m[x]);
    n_wait[x]--;
    if (n_wait[x]==0)
        sem_post(&b[x]);
    sem_post(&m[x]);
}

void signal_ch(int y){
    sem_wait(&b[y]);
    sem_wait(&m[y]);
    for (int i=0; i<n_wait[y]; i++){
        sem_post(&sync[y]);
    }
    sem_post(&m[y]);
}

```

## Exam 2022/02/10 - Ex 8 (4.0 points)

### Italiano

Dato il seguente grafo di precedenza, realizzarlo utilizzando il minimo numero possibile di semafori. Si utilizzino le primitive semaforiche implementate tramite `init`, `signal`, `wait` e `destroy` realizzate in pseudo-codice.

I processi rappresentati devono essere processi ciclici (con corpo del tipo `while(1)`).

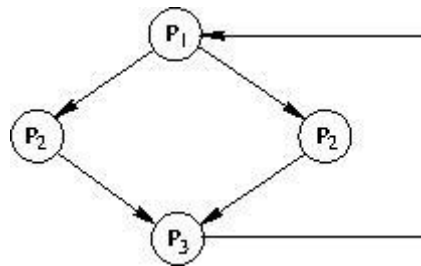
Si osservi che il grafo prevede l'esecuzione di 3 processi {P1, P2, P3} e che esistono due istanze (ovviamente identiche) del processo P2 ciascuna delle quali deve essere eseguita una volta (è errato eseguire due volte la stessa istanza per ogni ciclo principale).

### English

Given the following precedence graph, implement it using the minimum number of semaphores. Use semaphore primitives implemented through the pseudo-code functions `init`, `signal`, `wait`, and `destroy`.

The processes represented must be cyclical processes (with body of the type `while(1)`).

Note that the graph involves the execution of 3 processes {P1, P2, P3} and that there are two instances (obviously identical) of P2 each of which must be executed once (it is incorrect to run the same instance twice per main cycle).



Risposta: **Answer:**

Initialization:

```
init (s1, 1);
init (s2, 0);
init (s3, 0);
init (s, 0);
n = 0;
```

Termination:

```
destroy (s1);
..
destroy (s);
```

P1

```
while (1) {
    wait (s1);
    printf ("P1\n");
    signal (s2);
}
```

P2

```
while (1) {
    wait (s2);
    n++;
    if (n==1) {
        signal (s2);
    } else {
        n = 0;
        signal (s);
        signal (s);
    }
}
```

```

    }
    wait (s);
    printf ("P2\n");
    signal (s3);
}

P3
while (1) {
    wait (s3);
    wait (s3);
    printf ("P3\n");
    signal (s1);
}

```

### Exam 2021/01/29 - Ex 1 (3.0 points)

#### Italiano

Un programma concorrente è costituito da un unico processo di nome P1 (e funzione P1()), di tipo ciclico, di cui sono presenti 2 istanze.

Il comportamento del programma è il seguente:

- All'inizio le 2 istanze del processo P1 sono eseguite in parallelo.
- Quando entrambe le istanze di P1 hanno finito di eseguire il proprio codice (quello presente nella funzione P1()), entrambe le istanze di P1 sono eseguite nuovamente.

Si indichino quali dei seguenti codici sono corretti. Si osservi che risposte errate implicano una penalità nel punteggio finale.

#### English

A concurrent program is composed of a single process called P1 (and a function P1()). The process, and consequently the function, are cyclic. There are 2 instances of process P1.

The behavior of the program is as follows:

- At the beginning, the 2 instances of process P1 are executed in parallel.
- When both instances of P1 have finished executing their code (i.e., the code that is present in function P1()), both instances of P1 are executed again.

Indicate which of the following codes are correct. Note that wrong answers imply a penalty in the final score

Scegli UNA SOLA alternativa: Choose JUST ONE option:

1. ☐

```

int n=0;
init(s, 2);
init(m, 1);
init(b, 0);
while(1) {
    wait(s);
    P1();
    wait(m);
    n++;
    if (n==2) {
        signal(s);
        signal(s);
        n=0;
        signal(b);
    } else {
        signal(m);
        wait(b);
    }
}

```
2. ☒


```

int n=0;

```



```

init(s, 2);
init(m, 1);
init(b, 0);
while(1) {
    wait(s);
    P1();
    wait(m);
    n++;
    if (n==2) {
        signal(s);
        signal(s);
        n=0;
        signal(b);
        signal(m);
    } else {
        signal(m);
        wait(b);
    }
}
}
3.  int n=0;
init(s, 2);
init(m, 1);
while(1) {
    wait(s);
    P1();
    wait(m);
    n++;
    if (n==2) {
        signal(s);
        signal(s);
        n=0;
    }
    signal(m);
}

```

### Exam 2021/01/29 - Ex 14 (5.0 points)

#### Italiano

Un numero imprecisato di thread compete per l'utilizzo di un insieme di  $m$  risorse equivalenti fra loro e il cui numero è noto a priori.

Ogni thread esegue la funzione

```
int request(void)
```

per richiedere una risorsa. La funzione deve restituire l'indice  $i$  della risorsa assegnata al thread chiamante (dove  $i$  è un numero incluso tra 0 a  $m-1$ ). La politica di assegnazione deve essere decisa dalla funzione request, la quale deve anche assicurare che ogni risorsa sia assegnata a un solo thread in ogni istante. La funzione request deve porre il chiamante in attesa nel caso in cui non vi sia nessuna risorsa immediatamente disponibile e risvegliarlo non appena una risorsa risulta assegnabile.

Ogni thread esegue la funzione

```
void release(int i)
```

per rilasciare la risorsa di indice  $i$  allocata in precedenza.

Si realizzino le funzioni request e release usando i semafori POSIX, definendone le relative variabili condivise.

Si ipotizzi che i thread non tentino di rilasciare risorse non ad essi allocate e che nessun thread (considerato individualmente) tenti di allocare più di  $m$  risorse senza rilasciarne alcuna.

#### English

An unspecified number of threads compete for the use of a set of  $m$  equivalent resources, whose number is known a priori.

Each thread executes the function

```
int request(void)
```

to request a resource. The function must return the index  $i$  of the resource assigned to the calling thread (where  $i$  is a number between 0 to  $m-1$ ). The resources allocation policy must be decided by the request function, which must also ensure that each resource is assigned to only one thread at a time. The request function must block the caller if there is no resource immediately available, and wake it up as soon as a resource becomes available again. Each thread executes the function

```
void release(int i)
```

to release a previously allocated resource with index  $i$ .

Implement the request and release functions using POSIX semaphores, defining the related shared variables. Assume that threads do not attempt to release resources not allocated to them and that no thread (considered individually) attempts to allocate more than  $m$  resources without releasing any.

Risposta: **Answer:**

```
#define M ...
```

```
int resources[M] = {0}; /* Vector of resources: 1 occupied, 0 free. Initially all zeros. */
```

```
sem_t empty, mutex;
```

```
sem_init(&empty, M); /* Initially M resources available */
```

```
sem_init(&mutex, 1); /* Mutex to access resources shared vector */
```

```
int request(void){
```

```
    int i;
```

```
    sem_wait(&empty); /* If no resource available, it block the thread */
```

```
    sem_wait(&mutex);
```

```
    for(i=0; i<M; i++){ /* Search for a free resource */
```

```
        if (resources[i] == 0) {
```

```
            resources[i] = 1;
```

```
            break;
```

```
        }
```

```
    }
```

```
    sem_post(&mutex);
```

```
    return i;
```

```
}
```

```
void release(int i){
```

```
    sem_wait(&mutex);
```

```
    resources[i] = 0; /* Set the resource with index i as free */
```

```
    sem_post(&mutex);
```

```
    sem_post(&empty); /* Release the resource */
```

```
}
```

## Exam 2021/06/18 - Ex 13 (5.0 points)

### Italiano

In un sistema esistono due thread: uno di tipo A e uno di tipo B. Il thread di tipo A può eseguire la funzione `funz_a()`, mentre il thread di tipo B può eseguire la funzione `funz_b()`.

Il thread che esegue per primo una delle due funzioni è bloccato, mentre il secondo (quello che esegue l'altra funzione) non è bloccato e sblocca il processo precedentemente bloccato dall'altra funzione.

Tali funzioni possono essere riutilizzate dai thread A e B un numero indefinito di volte.

Si realizzino le funzioni `funz_a()` e `funz_b()` con il numero minimo di semafori POSIX, definendone le relative variabili condivise e i valori di inizializzazione dei semafori.

Esempio:

- A chiama la funzione funz\_a() e si blocca
- B chiama la funzione funz\_b() che sblocca A
- B chiama la funzione funz\_b() e si blocca
- A chiama la funzione funz\_a() che sblocca B

### English

In a system there are two threads: one of type A and one of type B. The type A thread can call the funz\_a() function, while the B-type thread can call the funz\_b() function.

The thread that performs one of the two functions first is blocked, while the second (the one running the other function) is not blocked and unlocks the thread previously blocked by the other function.

These functions can be reused by threads A and B an indefinite number of times.

Realize the funz\_a() and funz\_b() functions with the minimum number of POSIX semaphores, defining their shared variables and semaphore initialization values.

Example:

- A calls the function funz\_a() and it blocks
- B calls the function funz\_b() and it unlocks A
- B calls the function funz\_b() and it blocks
- A calls the function funz\_a() and unlocks B

Risposta: [Answer:](#)

```
// Solution 1
init(m, 1);
init(s, 0);
int flag=0;
funz_a() {
    wait(m);
    if (flag==0) {
        flag=1;
        signal(m);
        wait(s);
    } else {
        flag=0;
        signal(m);
        signal(s);
    }
}
funz_b() {
    wait(m);
    if (flag==0) {
        flag=1;
        signal(m);
        wait(s);
    } else {
        flag=0;
        signal(m);
        signal(s);
    }
}

// Solution 2
init(a, 0);
init(b, 0);
funz_a() {
    signal(b);
    wait(a);
}
```

```
funz_b() {  
    signal(a);  
    wait(b);  
}
```