P10

Uvod u RTOS, 2. deo

ArdOS – kompakt RTOS

- Kernel (bez taskova): ispod 2k flash-a i manje od 200 B RAM-a
- Preemptive scheduling sa prioritetima
- Kooperativni scheduling
- Sleep funkcija koja oslobađa MCU za druge taskove
- Binarni i brojački semafori
- Mutex
- FIFO i prioritetni messaging
- Konfigurabilan radi štednje memorije

Context switching

1) Šta je to kontekst?
2) Funkcija OS – ContextSwitcher()
3) Dodatna memorija za čuvanje konteksta za task

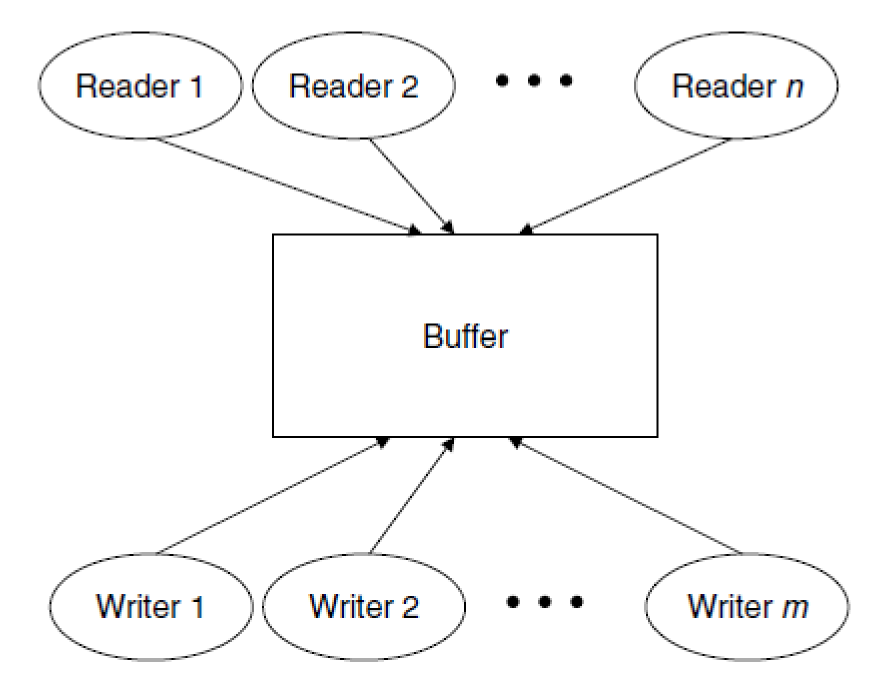
Sinhronizacija i komunikacija među taskovima

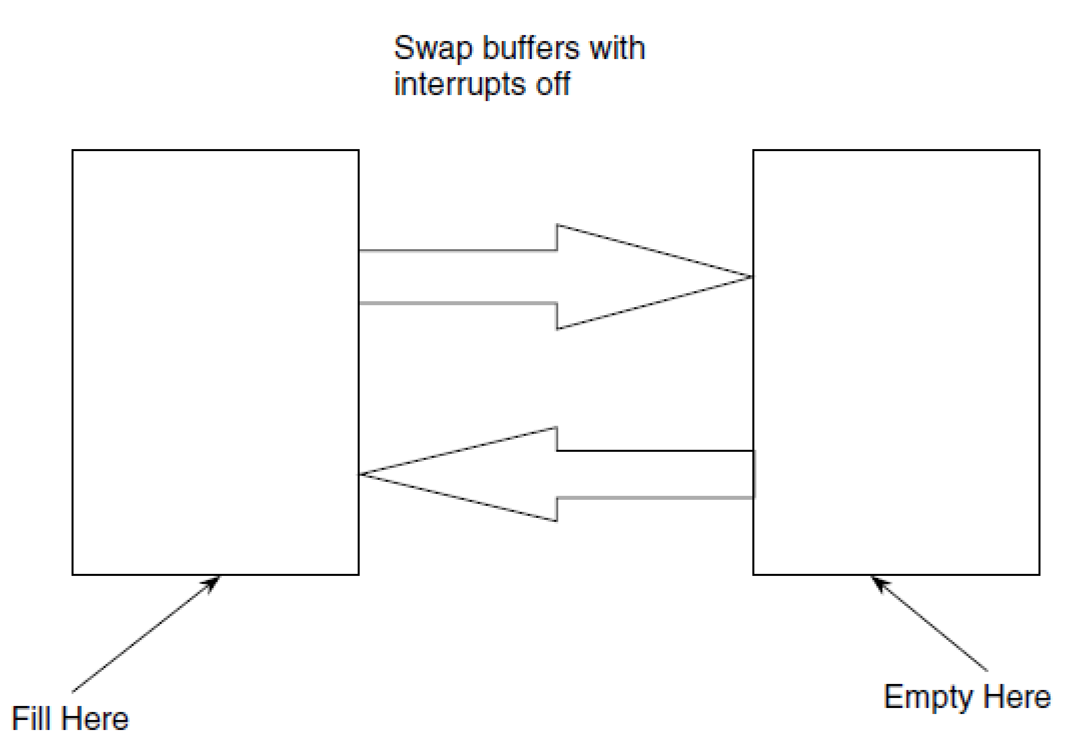
1)Poruke
2)Semafori (binarni i brojački)
3)Mutexi

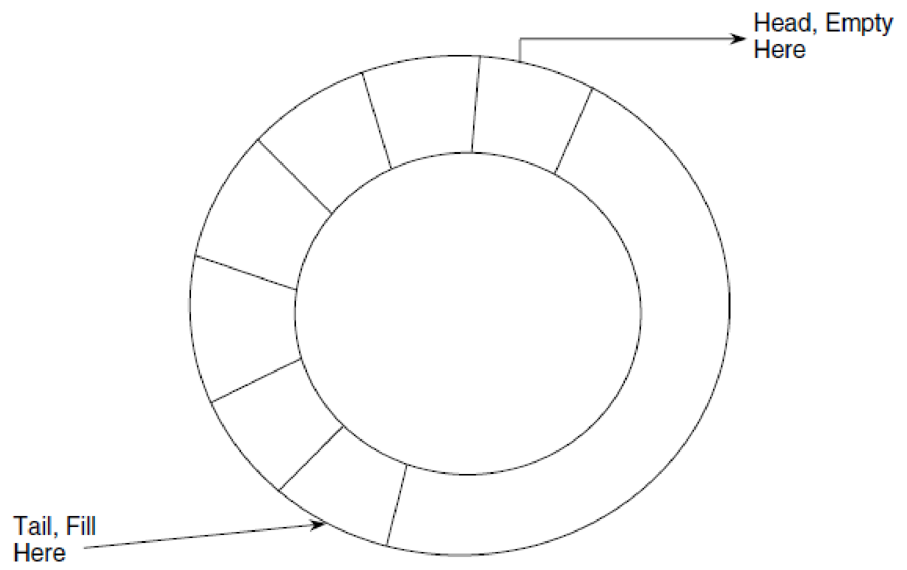
Poruke

1) Globalne varijable
2) Baferi (običan, *swap*, *ring*)
3) *Mailbox*
4) Red čekanja (*queue*)

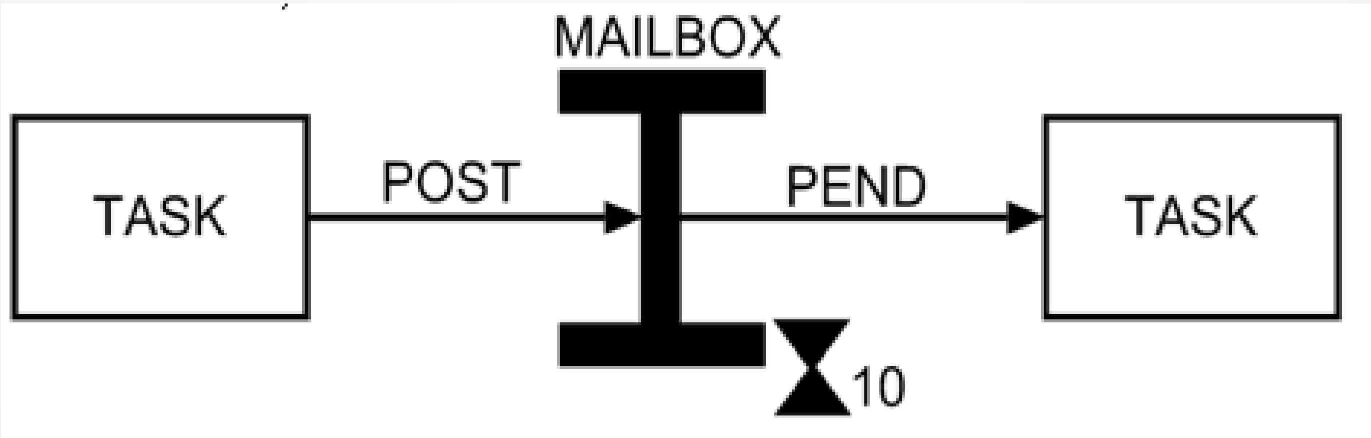
Baferi


Reader 1
Reader 2
• • •
Reader *n*
Buffer
Writer 1
Writer 2
• • •
Writer *m*

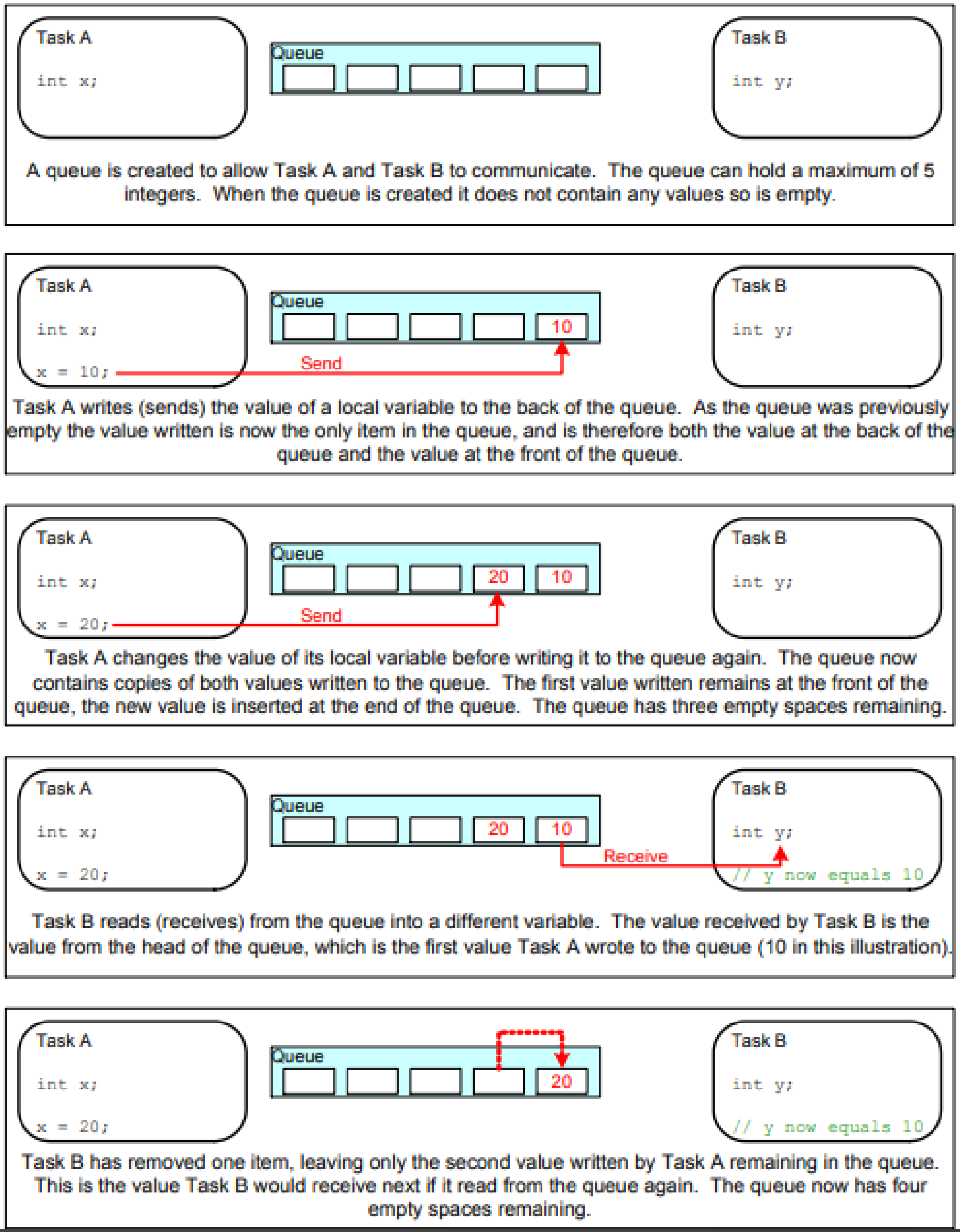

Swap buffers with
interrupts off
Fill Here
Empty Here


Head, Empty
Here
Tail, Fill
Here

Mailbox



Queue



A queue is created to allow Task A and Task B to communicate. The queue can hold a maximum of 5 integers. When the queue is created it does not contain any values so is empty.

Task A writes (sends) the value of a local variable to the back of the queue. As the queue was previously empty the value written is now the only item in the queue, and is therefore both the value at the back of the queue and the value at the front of the queue.

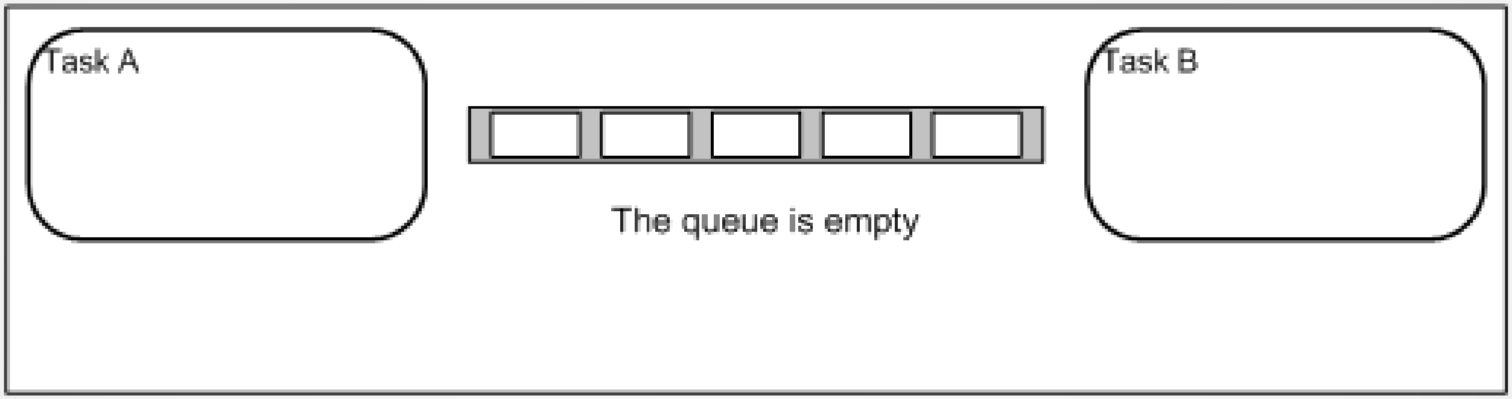
Task A changes the value of its local variable before writing it to the queue again. The queue now contains copies of both values written to the queue. The first value written remains at the front of the queue, the new value is inserted at the end of the queue. The queue has three empty spaces remaining.

Task B reads (receives) from the queue into a different variable. The value received by Task B is the value from the head of the queue, which is the first value Task A wrote to the queue (10 in this illustration).

Task B has removed one item, leaving only the second value written by Task A remaining in the queue. This is the value Task B would receive next if it read from the queue again. The queue now has four empty spaces remaining.

https://www.freertos.org/Embedded-RTOS-Queues.html

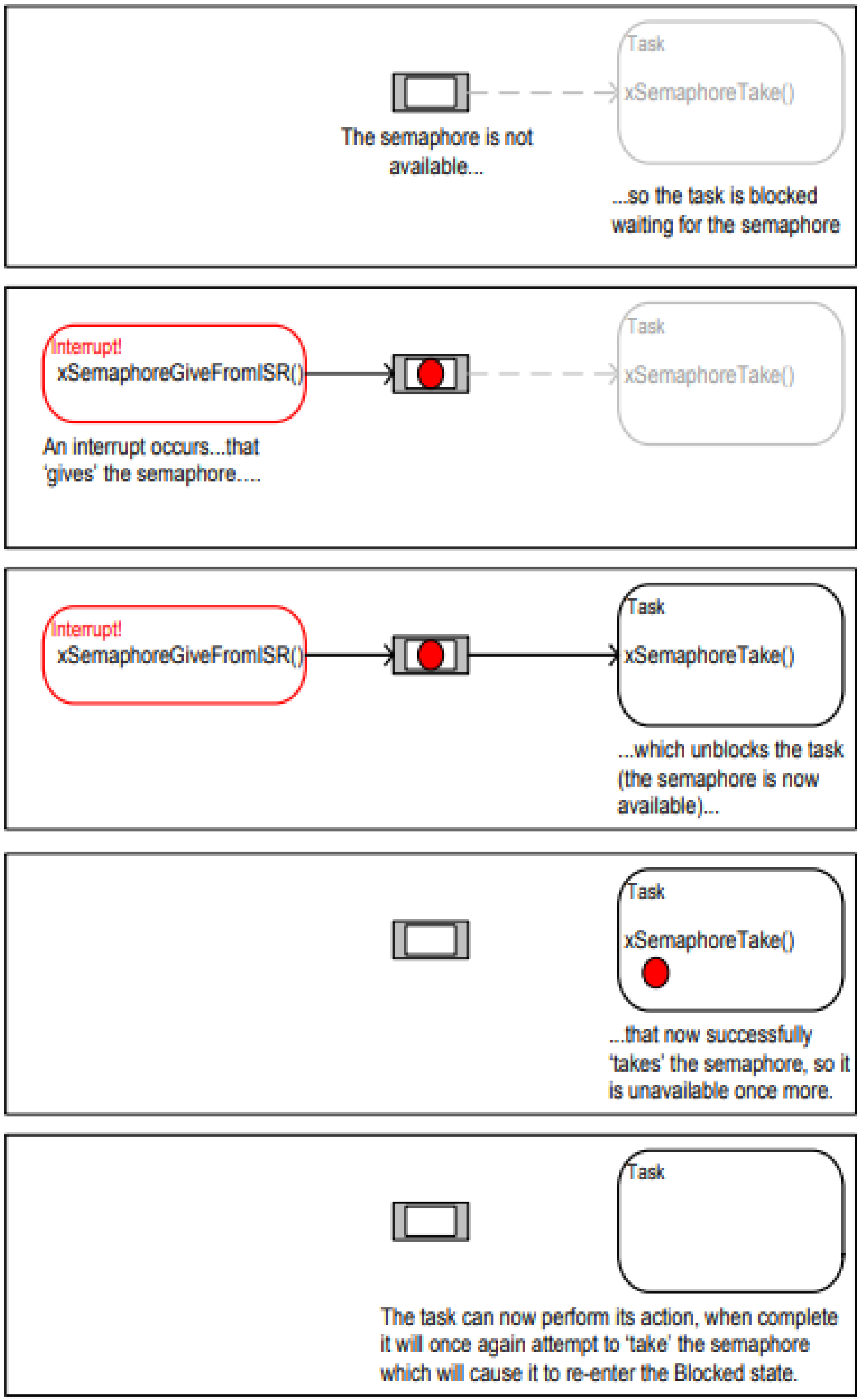
Queue



The queue is empty

https://www.freertos.org/Embedded-RTOS-Queues.html

Semafori (binarni)



The semaphore is not available...

...so the task is blocked waiting for the semaphore

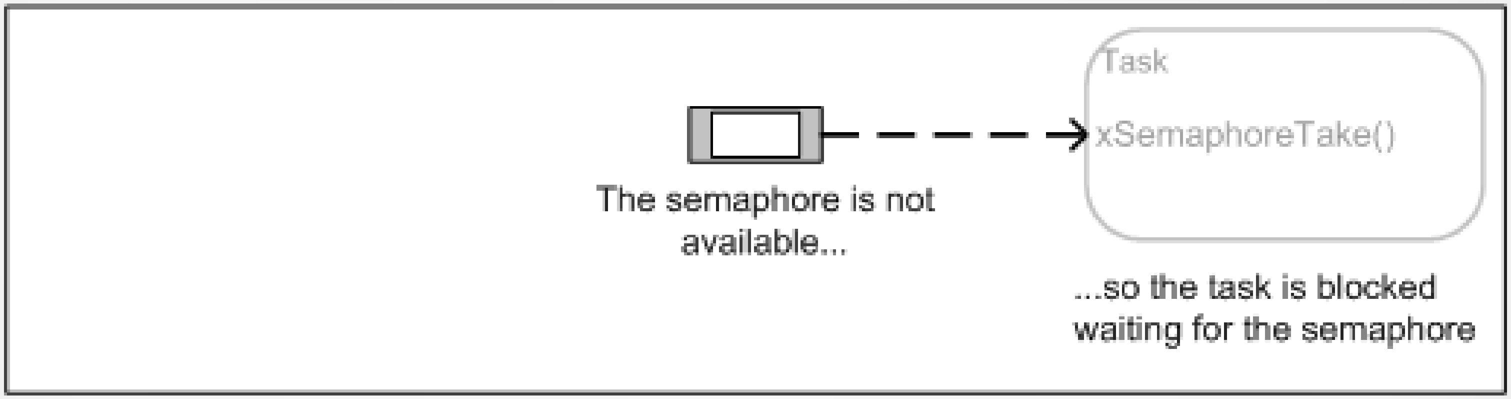
An interrupt occurs...that 'gives' the semaphore....

...which unblocks the task (the semaphore is now available)...

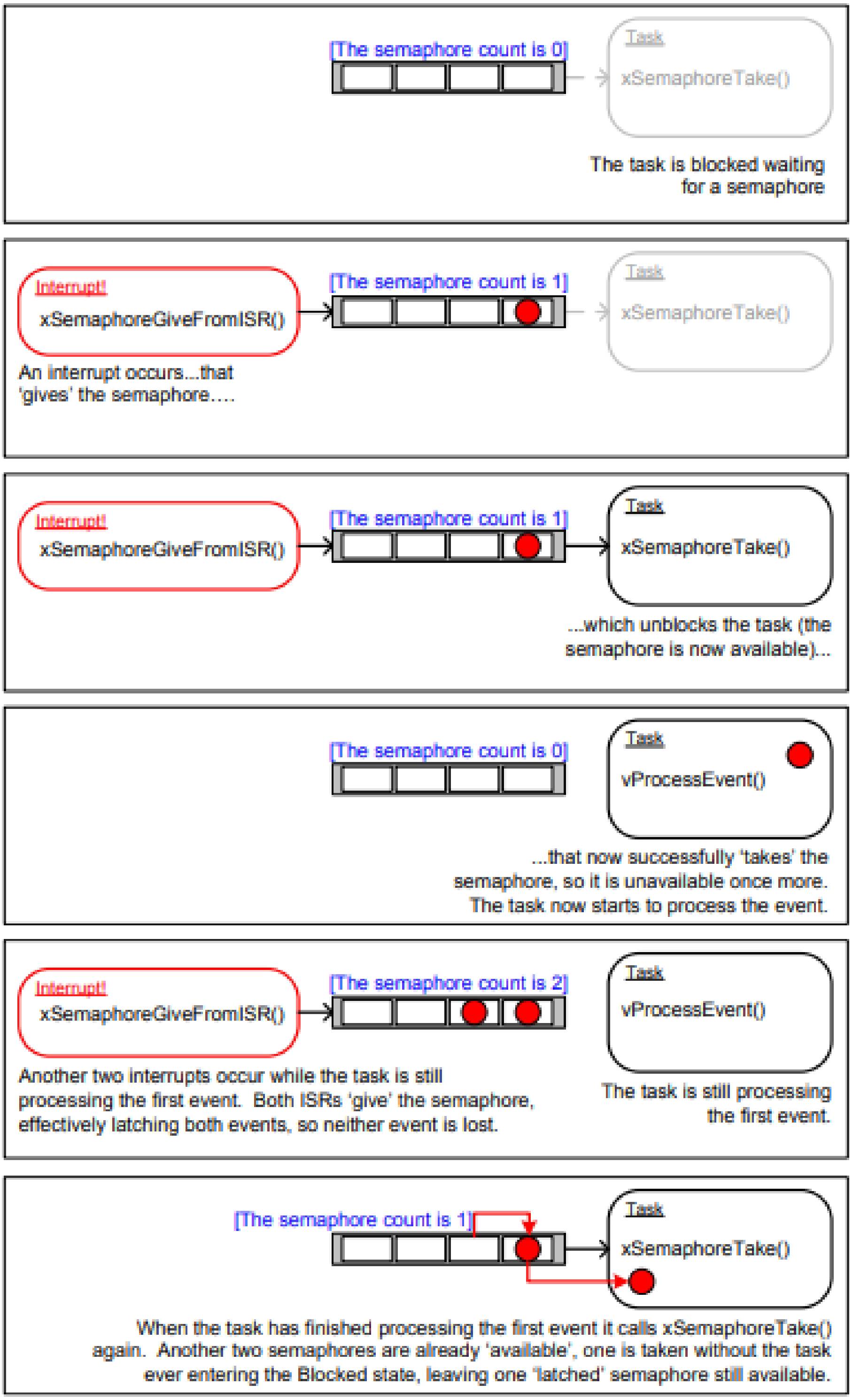
...that now successfully 'takes' the semaphore, so it is unavailable once more.

The task can now perform its action, when complete it will once again attempt to 'take' the semaphore which will cause it to re-enter the Blocked state.

Semafori (binarni)


Task
xSemaphoreTake()
The semaphore is not available...
...so the task is blocked waiting for the semaphore

Semafori (brojački)



The task is blocked waiting for a semaphore

An interrupt occurs...that 'gives' the semaphore....

...which unblocks the task (the semaphore is now available)...

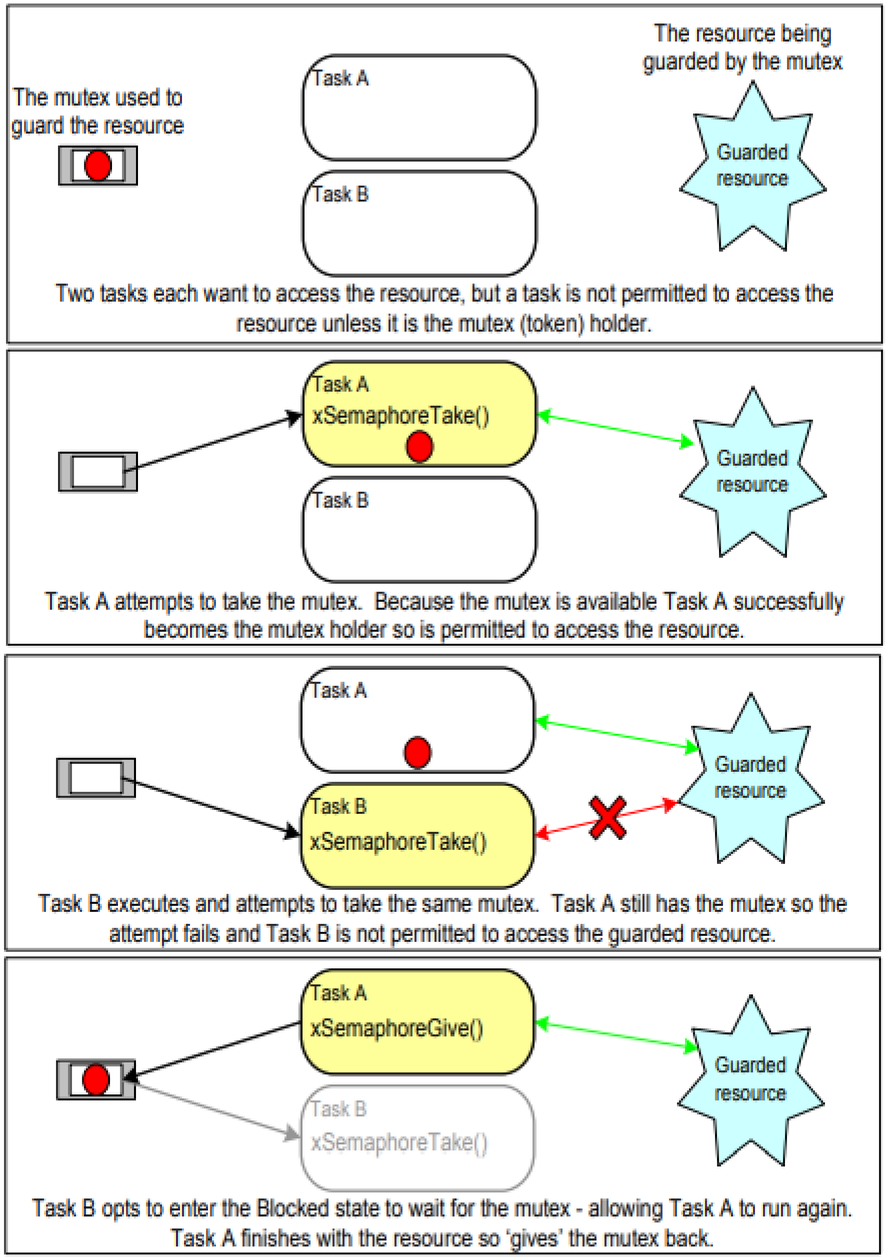
...that now successfully 'takes' the semaphore, so it is unavailable once more. The task now starts to process the event.

Another two interrupts occur while the task is still processing the first event. Both ISRs 'give' the semaphore, effectively latching both events, so neither event is lost.

The task is still processing the first event.

When the task has finished processing the first event it calls xSemaphoreTake() again. Another two semaphores are already 'available', one is taken without the task ever entering the Blocked state, leaving one 'latched' semaphore still available.

Mutex (binarni semafori sa prioritetom)


The mutex used to guard the resource
Task A
Task B
The resource being guarded by the mutex
Guarded resource

Two tasks each want to access the resource, but a task is not permitted to access the resource unless it is the mutex (token) holder.

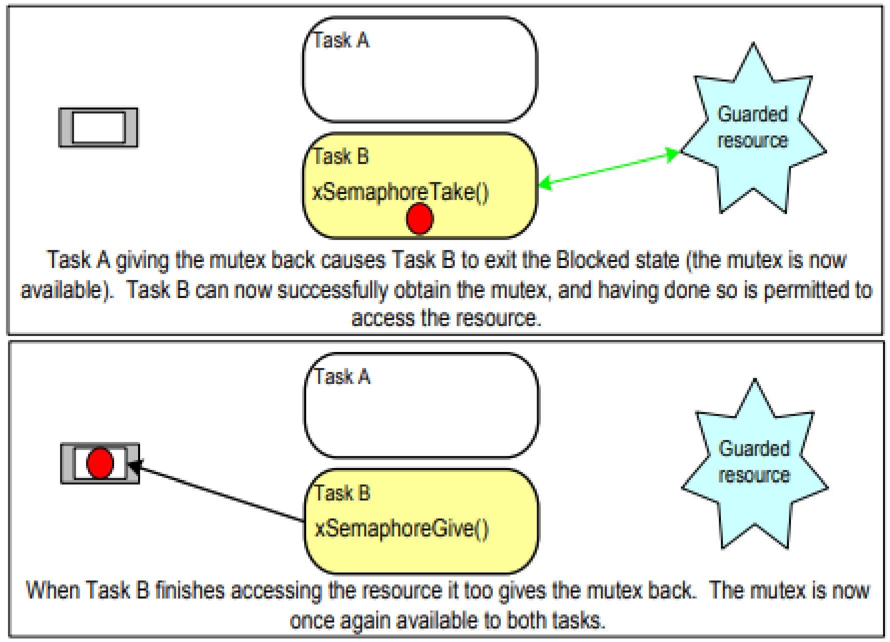
Task A attempts to take the mutex. Because the mutex is available Task A successfully becomes the mutex holder so is permitted to access the resource.

Task B executes and attempts to take the same mutex. Task A still has the mutex so the attempt fails and Task B is not permitted to access the guarded resource.

Task B opts to enter the Blocked state to wait for the mutex - allowing Task A to run again. Task A finishes with the resource so 'gives' the mutex back.

MUTEX=mutual exlusion

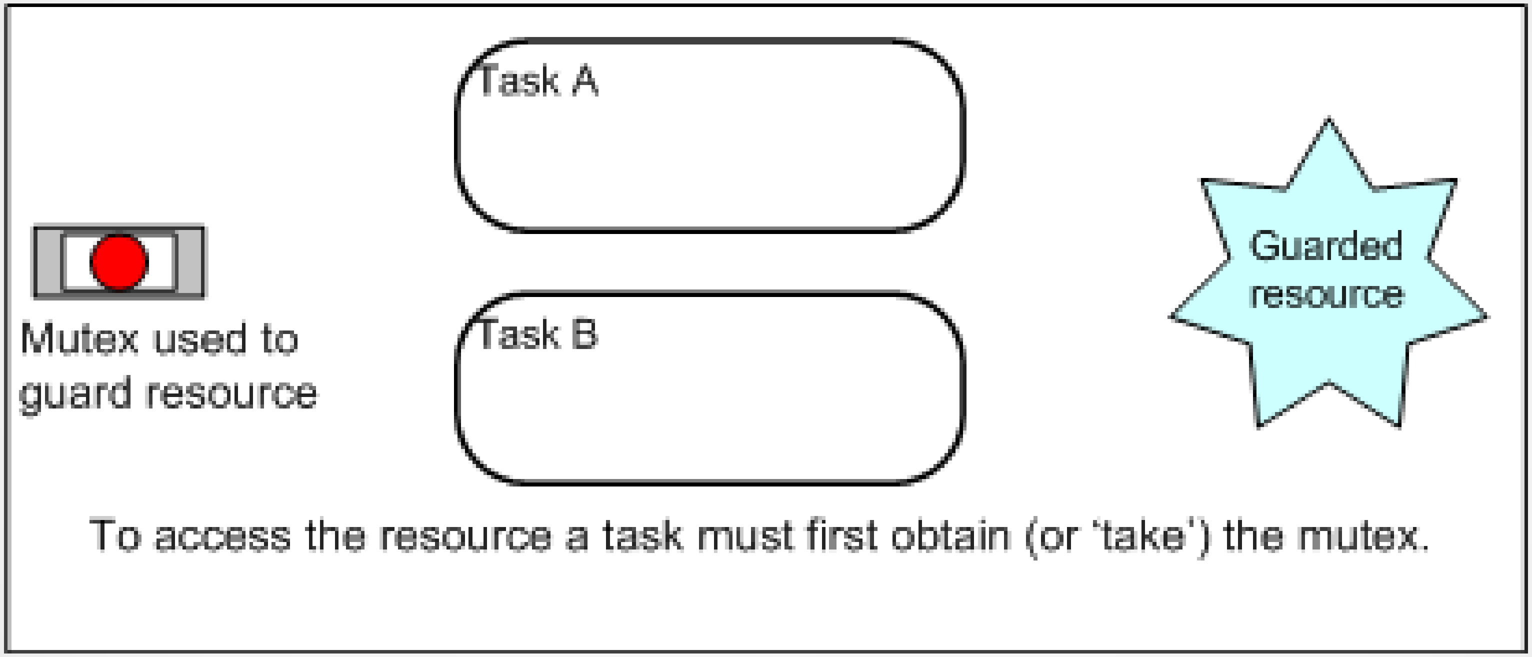
Kontrola pristupa deljenom resursu



Task A giving the mutex back causes Task B to exit the Blocked state (the mutex is now available). Task B can now successfully obtain the mutex, and having done so is permitted to access the resource.

When Task B finishes accessing the resource it too gives the mutex back. The mutex is now once again available to both tasks.

Mutex



Task A

Task B

Mutex used to guard resource

Guarded resource

To access the resource a task must first obtain (or ‘take’) the mutex.

Preemptive RTOS

-nadgradnja kooperativnog dodatnim mogućnostima

-prelazi i u slučaju da nema poziva funkcija za promenu taska

Dodatni mehanizmi za prelaz

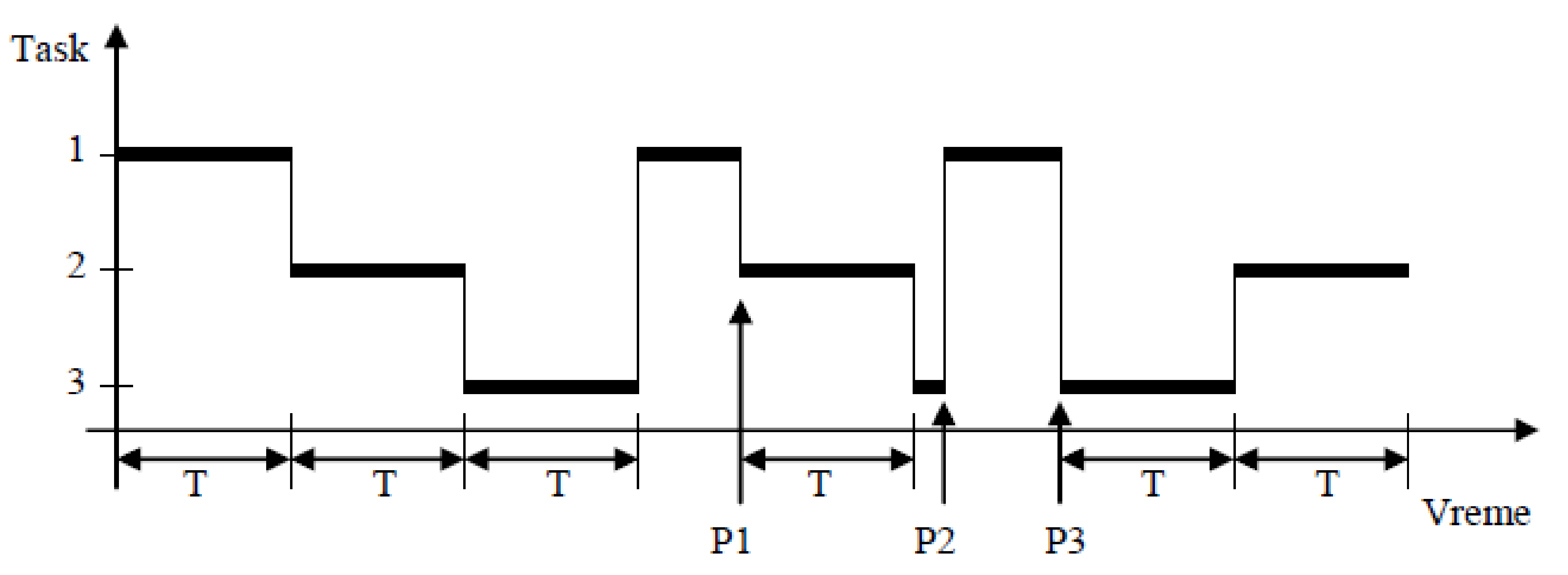
-Aktiviranje iz prekidne funkcije sistemskog tajmera

-Aktiviranje nakon nekog događaja koji prioritetni task dovodi u stanje *Ready*

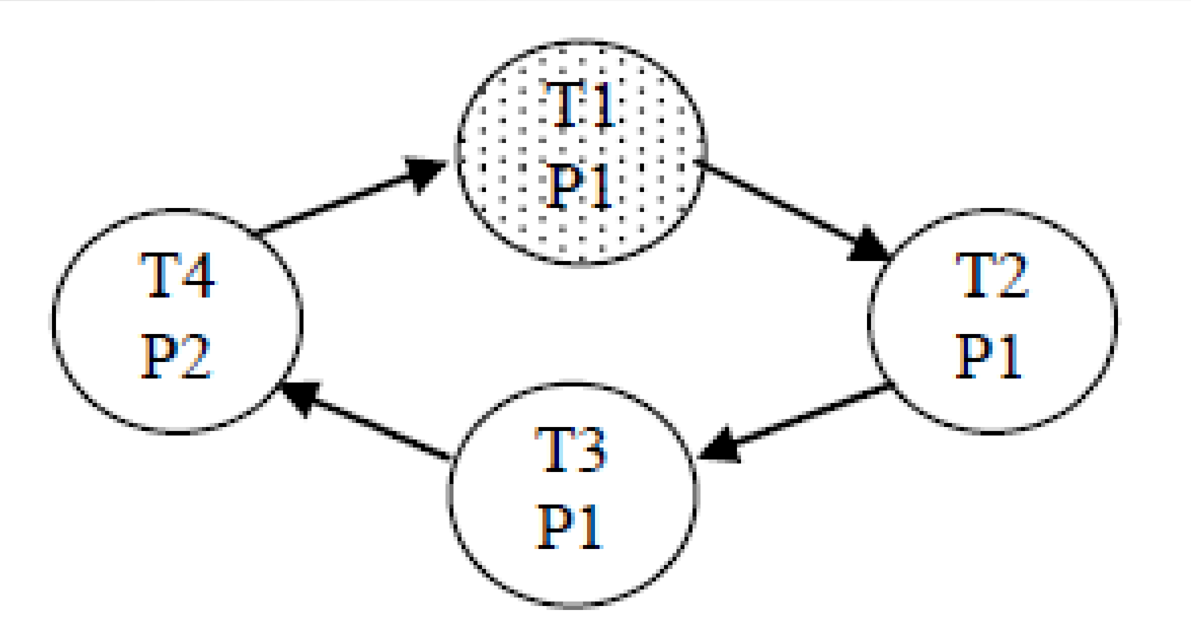
-Prioritet taskova

-*Timeout* taskova

Vremenska raspodela rada taskova



Promena taskova u slučaju različitih prioriteta

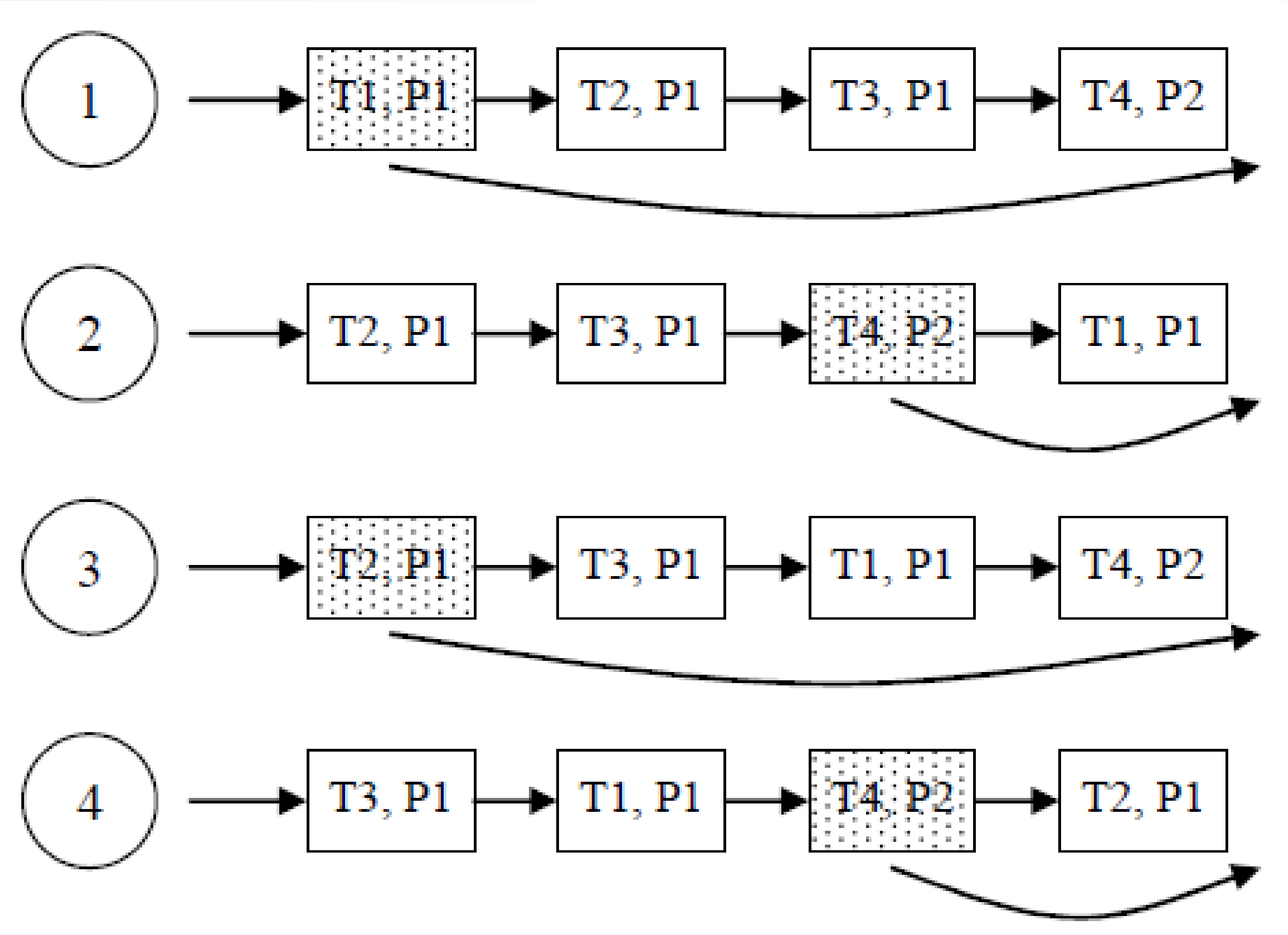


T1,T2 i T3- nižeg prioriteta i imaju timeout
T4 – višeg prioriteta i reaguje na ext. hardverske impulse

Problem?

Round Robin

Promena taskova po principu ulančane liste



Izbor taska kreće od početka ulančane liste poštujući prioritet

Task koji je završio ide na kraj liste

Osobine

1) Problem steka ne može da se izbegne jer se ne zna kada se poziva funkcija za promenu taska

2) Poseban stek za svaki task

3) Potrebno znatno više RAM-a

4) Obično se ne koristi kod sistema sa malo RAM-a

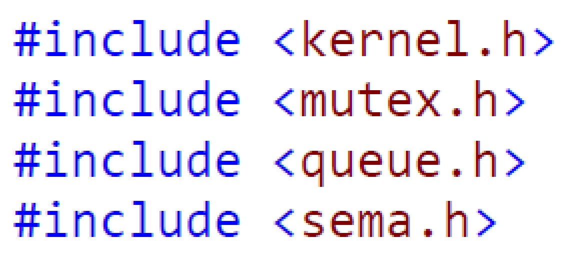
A, Ar, … ArdOS

Instalacija:

1) Download ArdOS-v09b zip fajla
2) U Arduino IDE: Sketch->Include Library->Add .ZIP Library
3) Nakon toga se u listi biblioteka u grupi ‘Contributed library’ pojavi ArdOS-v09b

Rad sa ArdOS:

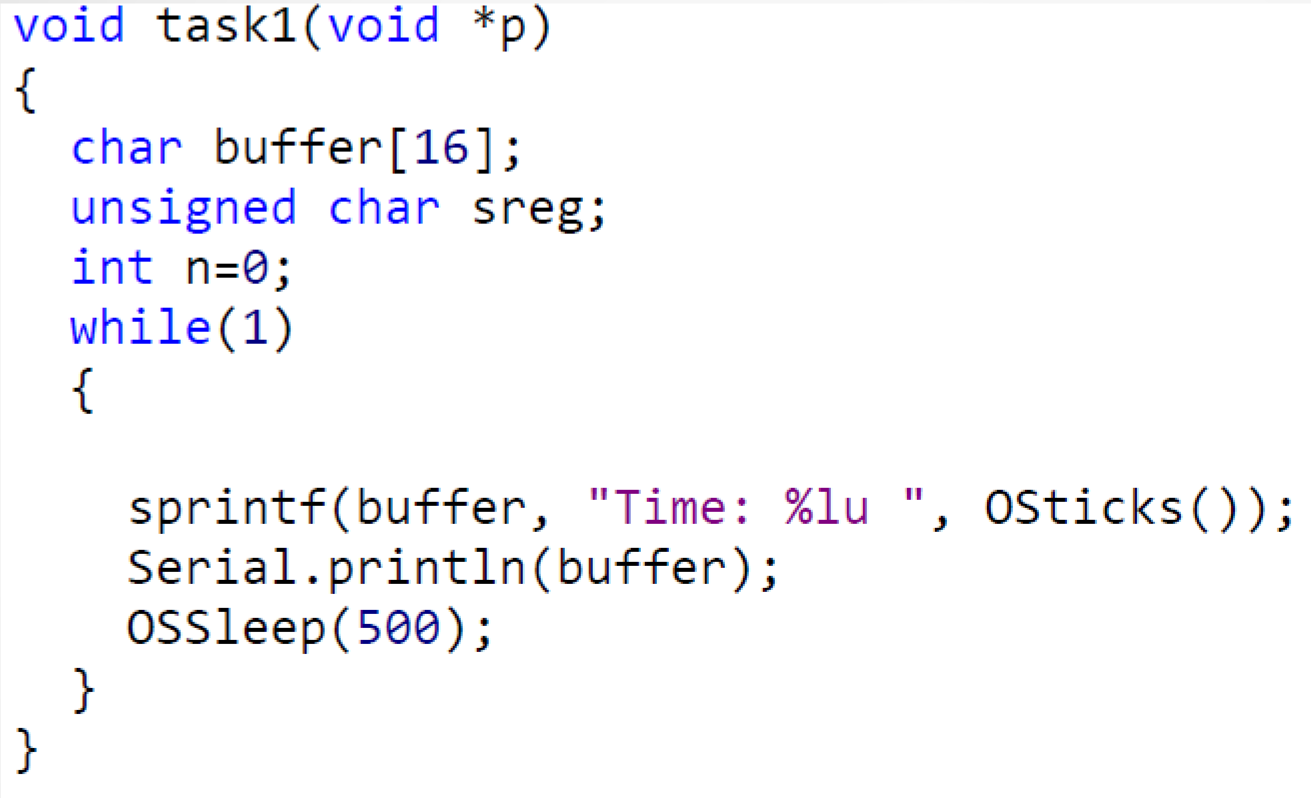
U praznom sketch-u uraditi Import Library -> ArdOS-v09b što automatski doda:



```
#include <kernel.h>
#include <mutex.h>
#include <queue.h>
#include <sema.h>
```

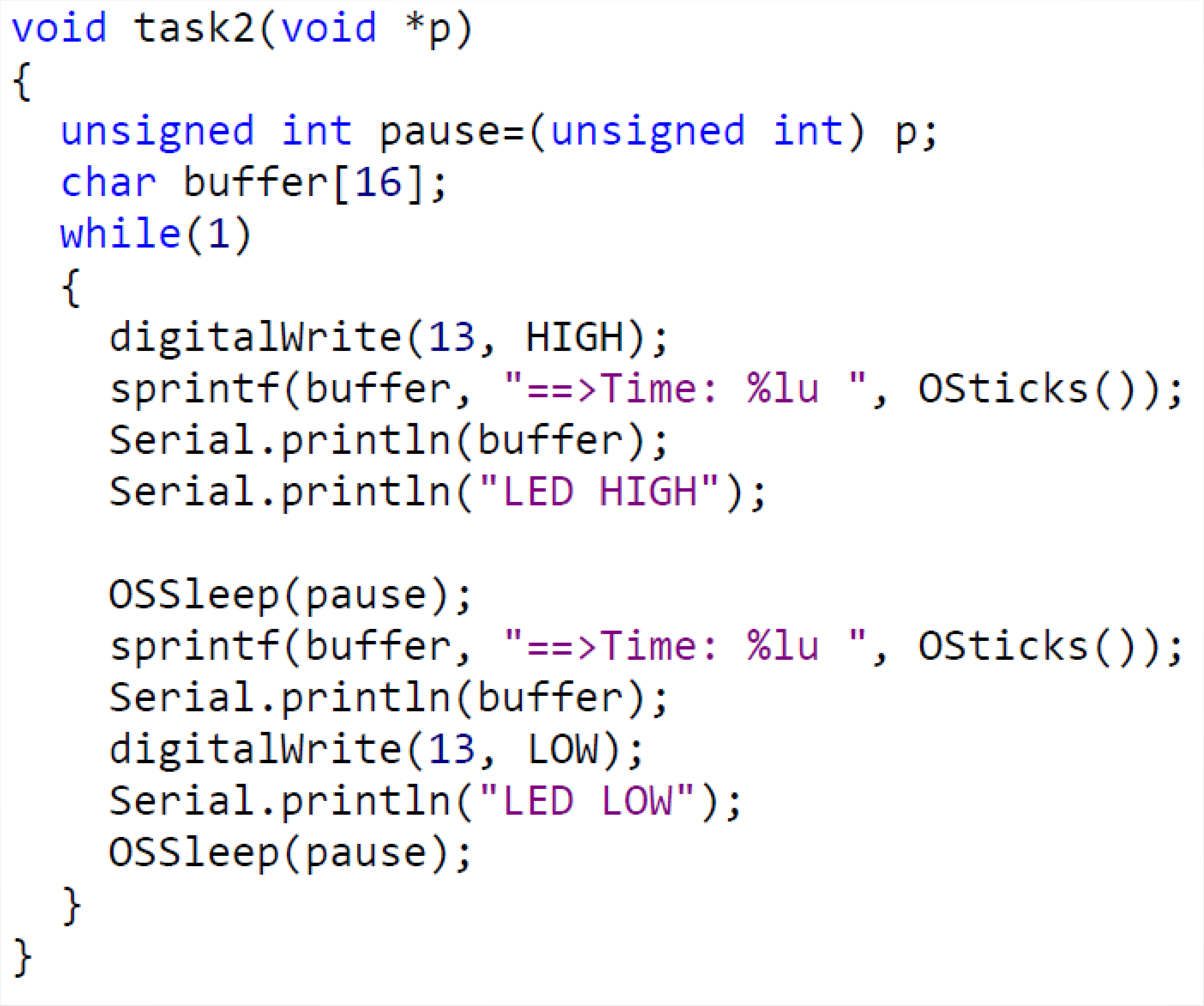
https://www.codeproject.com/Articles/834674/Stage-Getting-Started-With-ArdOS-for-Arduino#3.%20ArdOS

ArdOS



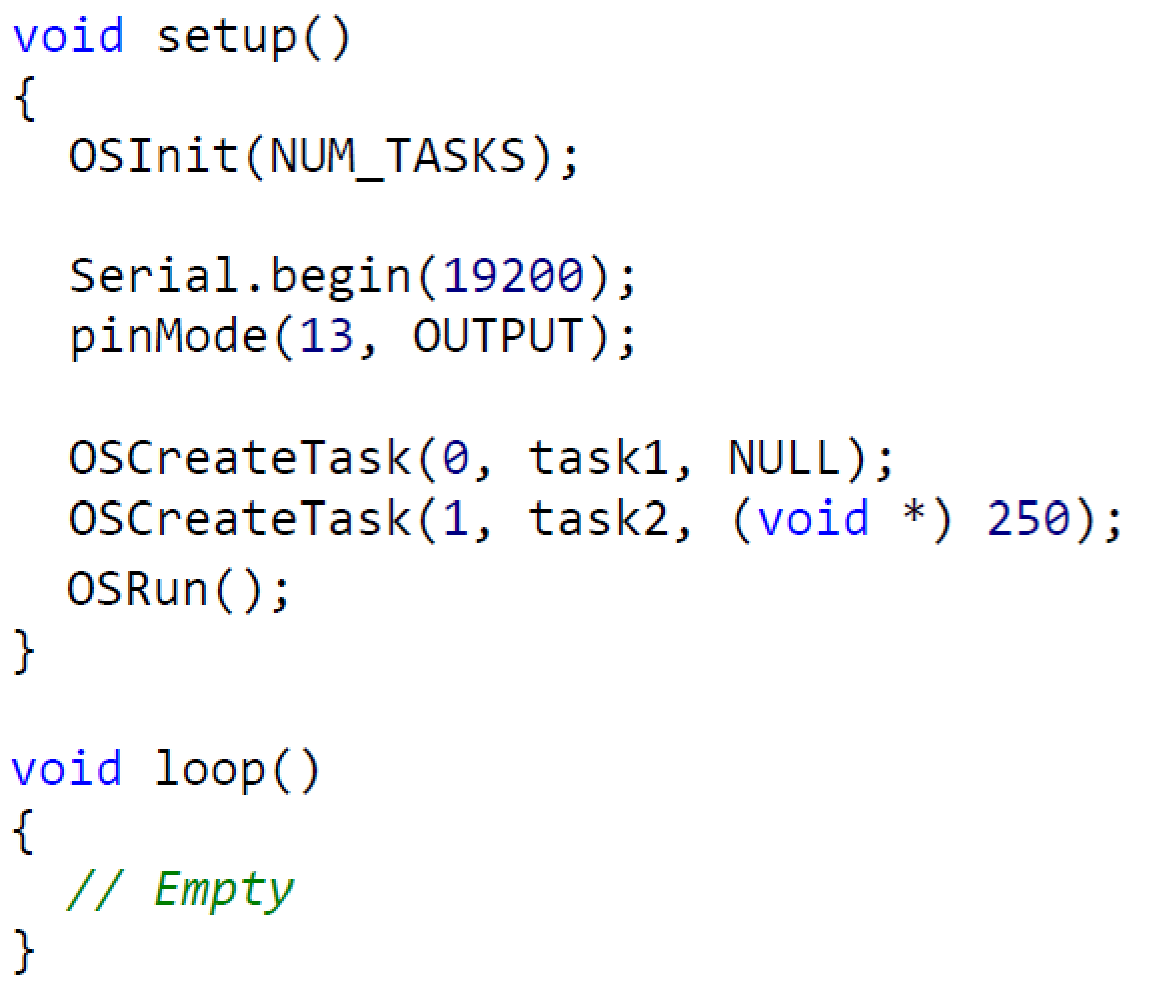
```
void task1(void *p)
{
  char buffer[16];
  unsigned char sreg;
  int n=0;
  while(1)
  {

    sprintf(buffer, "Time: %lu ", OSticks());
    Serial.println(buffer);
    OSSleep(500);
  }
}
```

ArdOS



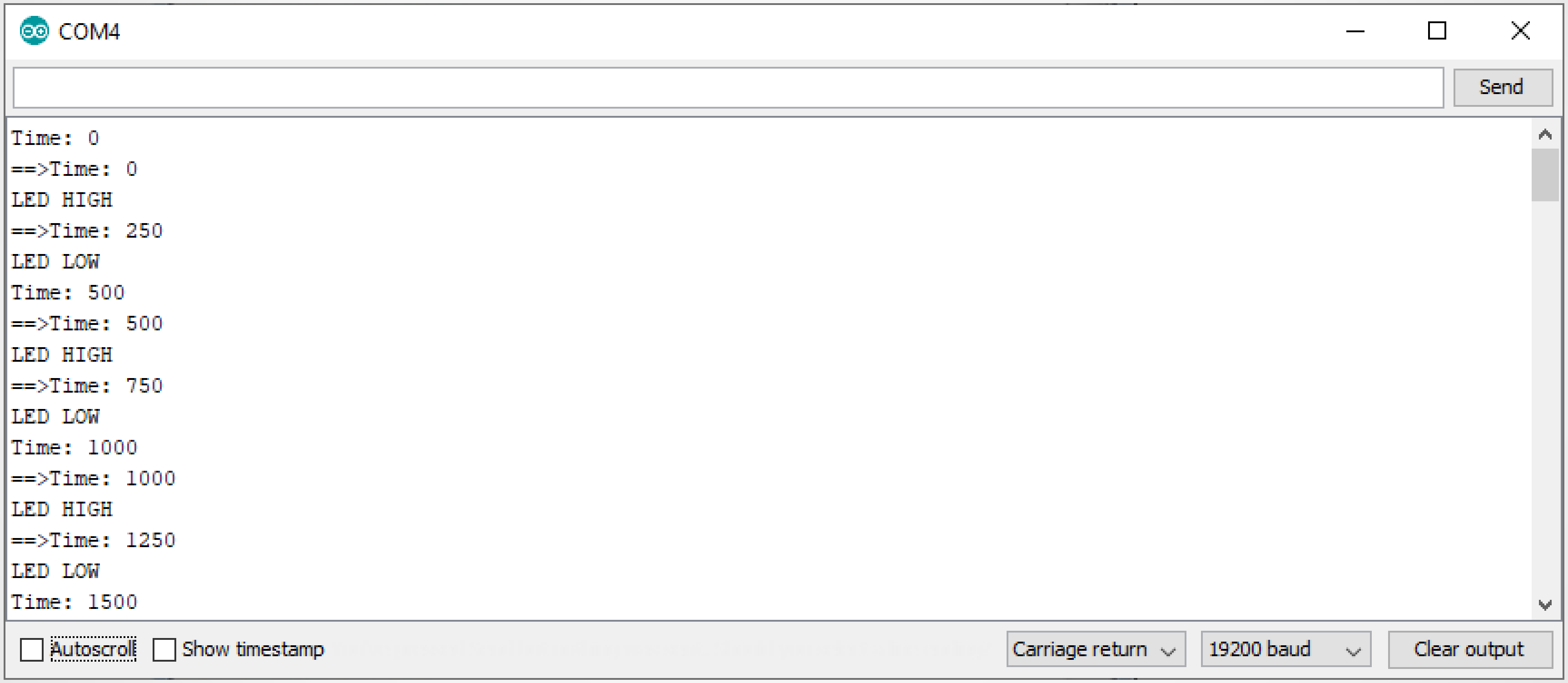
```
void task2(void *p)
{
  unsigned int pause=(unsigned int) p;
  char buffer[16];
  while(1)
  {
    digitalWrite(13, HIGH);
    sprintf(buffer, "==>Time: %lu ", OSticks());
    Serial.println(buffer);
    Serial.println("LED HIGH");

    OSSleep(pause);
    sprintf(buffer, "==>Time: %lu ", OSticks());
    Serial.println(buffer);
    digitalWrite(13, LOW);
    Serial.println("LED LOW");
    OSSleep(pause);
  }
}
```

ArdOS



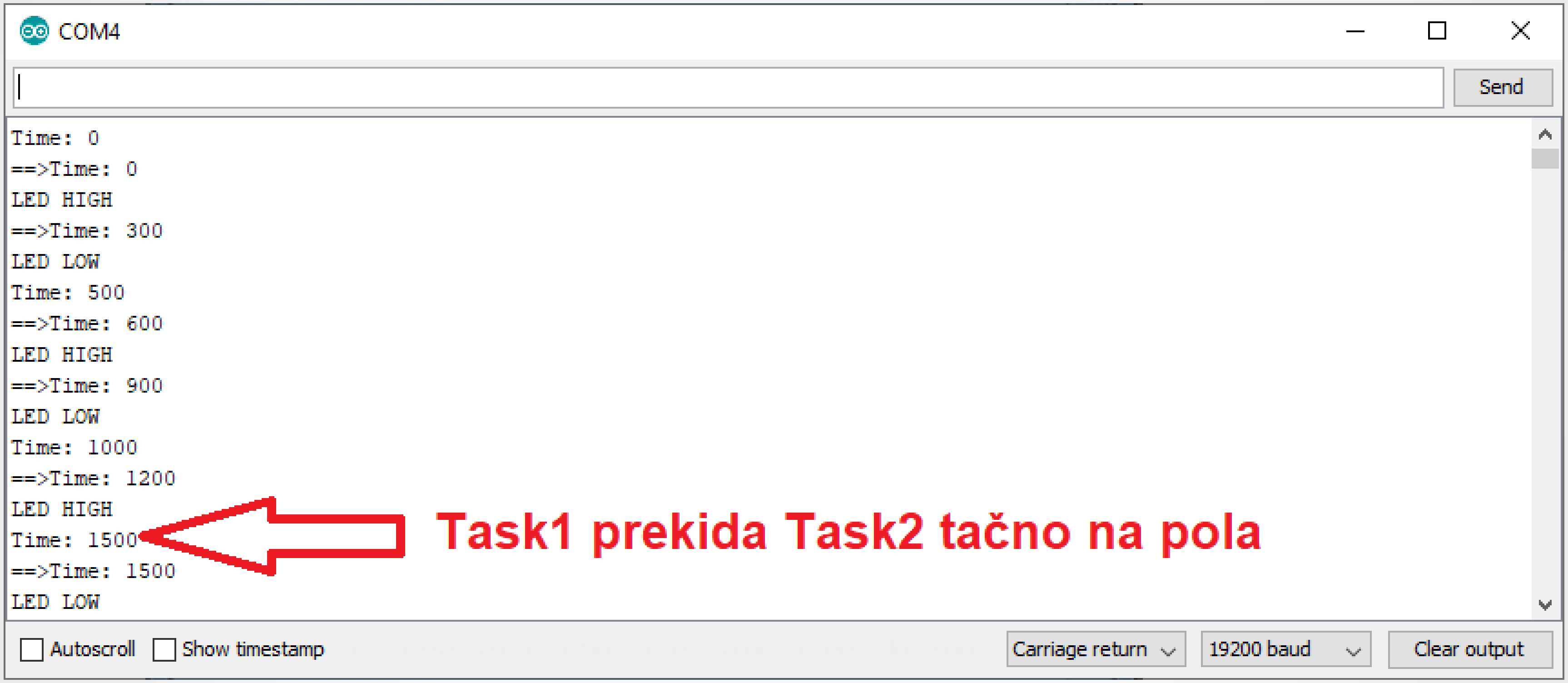
```
void setup()
{
  OSInit(NUM_TASKS);

  Serial.begin(19200);
  pinMode(13, OUTPUT);

  OSCreateTask(0, task1, NULL);
  OSCreateTask(1, task2, (void *) 250);
  OSRun();
}

void loop()
{
  // Empty
}
```

ArdOS (p=250)



```
Time: 0
==>Time: 0
LED HIGH
==>Time: 250
LED LOW
Time: 500
==>Time: 500
LED HIGH
==>Time: 750
LED LOW
Time: 1000
==>Time: 1000
LED HIGH
==>Time: 1250
LED LOW
Time: 1500
```

ArdOS (p=300)



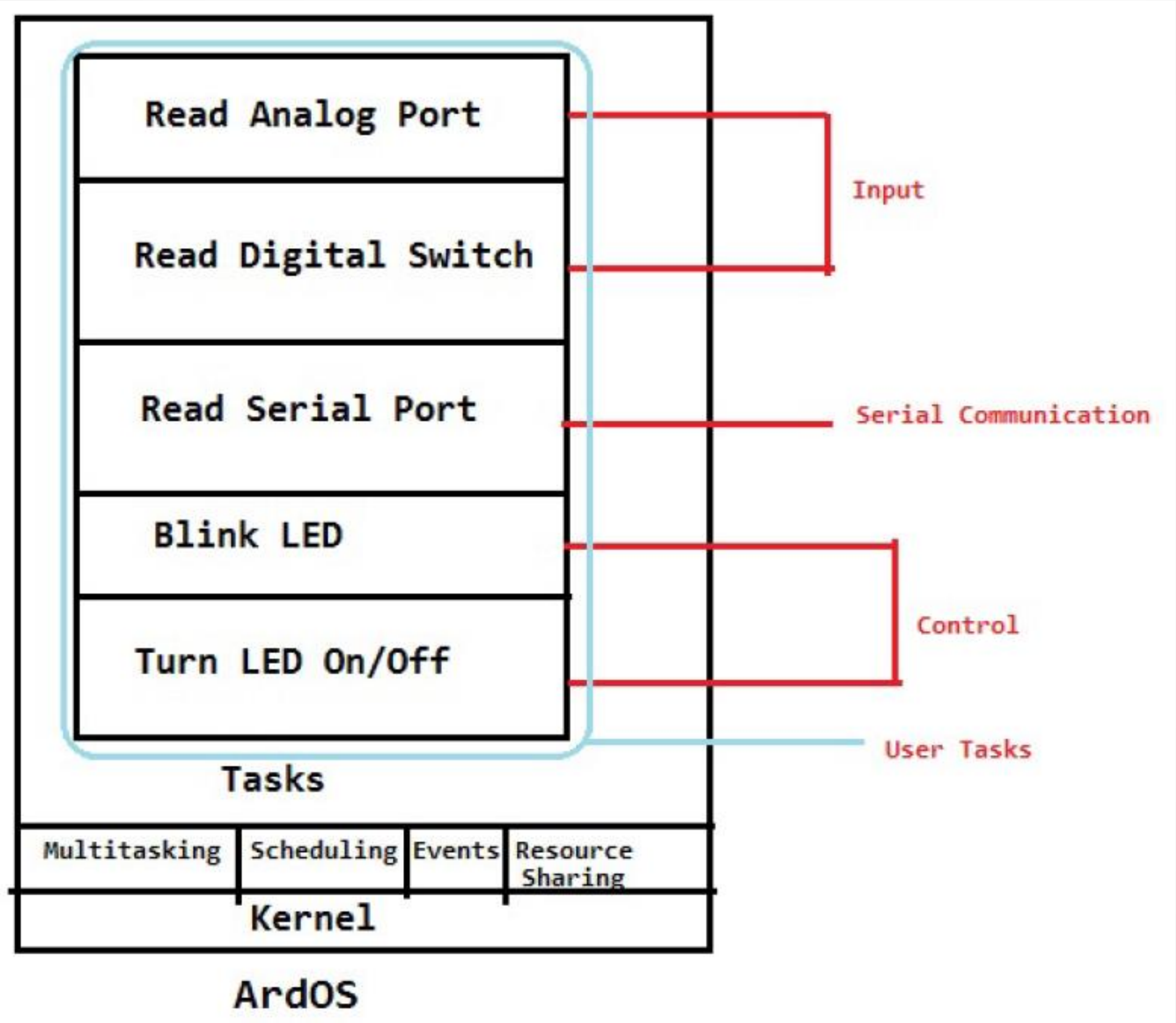
```
Time: 0
==>Time: 0
LED HIGH
==>Time: 300
LED LOW
Time: 500
==>Time: 600
LED HIGH
==>Time: 900
LED LOW
Time: 1000
==>Time: 1200
LED HIGH
Time: 1500
==>Time: 1500
LED LOW
```

Task1 prekida Task2 tačno na pola

Zaključak

Pažljivo projektovanje taskova!

Još malo ArdOS-a



HVALA NA PAŽNJI !