



OPERACIJSKI SUSTAVI

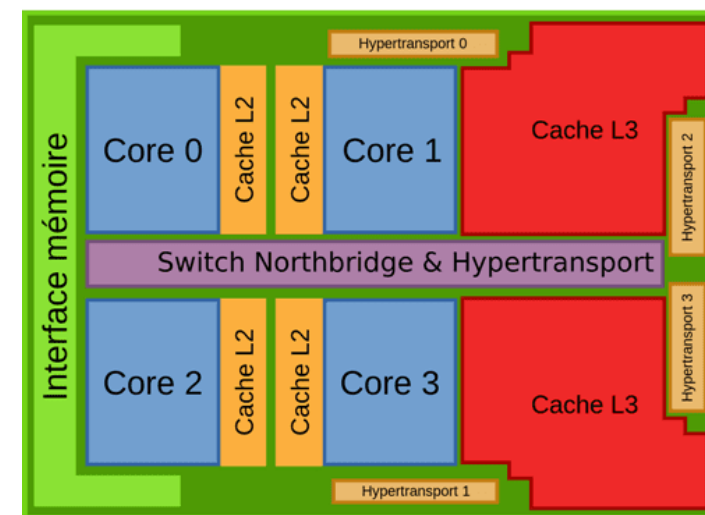
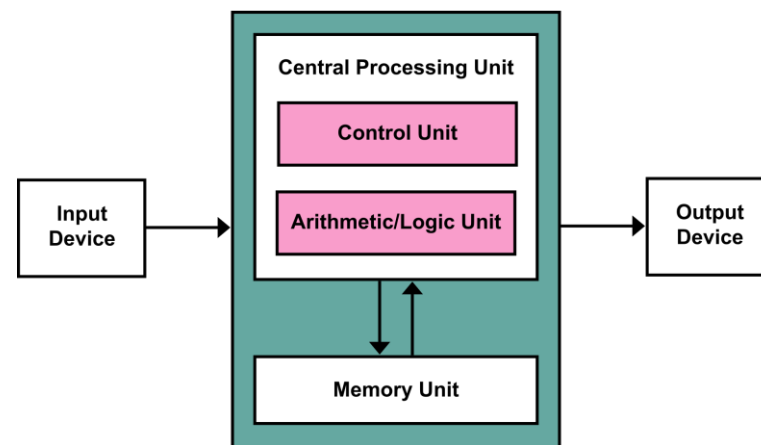
Procesi i dretve

Što ćemo danas raditi?

- Što je proces
- Što je dretva
- Primjeri u C/C++

Jezgra računala (Core)

- Jezgra = fizički procesor = CPU
- Danas skoro sva računala imaju barem **dvije** jezgre
- Procesor može imati jednu ili više **istih** jezgri
- Svaka jezgra radi brzinom od 1 GHz do 5 GHz (overclocking/turbo za više)
- **Zašto nema procesora sa jednom jako brzom jezgrom?**



Procesor sa 4 jezgre

Usporedba Intel procesora

Property	i3	i5	i7	i9
Max. Base clock	3.8 GHz	4.1 GHz	3.8 GHz	3.7 GHz
Max. Turbo boost rate	unsupported	unsupported	5.1 GHz	5.2 GHz
Single core turbo	up to 4.6 GHz	up to 4.8 GHz	up to 5.0 GHz	up to 5.1 GHz
Max cores / threads	4/8	6/12	8/16	10/20



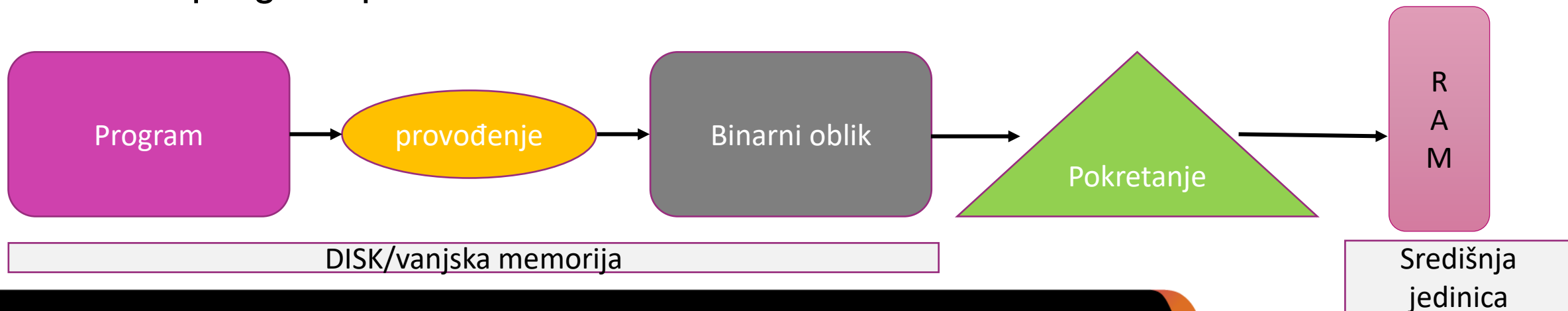
To nisu programske dretve/thread (već virtualni CPU)

Programi, procesi i dretve

- Program:
 - Skup uputa računalu što treba učiniti i kako to izvesti
 - Primjeri programa:
 - Programi koje koristi operativni sustav,
 - Text-procesori za pisanje teksta
 - Proračunske tablice
 - Baze podataka
 - Programi koji imaju određenu namjenu zovemo „Aplikacije” (npr. Web preglednik, MS Word...)
 - Programi su smješteni na disku u formatu koji se može izvršiti na računalu

Program

- Napisan u nekom programskom jeziku (C, Python, PHP, JavaScript...)
- Program se prevodi iz tekstualnog oblika u binarni oblik da se može izvršiti na računalu
 - Neki programi koriste „interpretere” te se provođenje radi samo kada se program pokrene

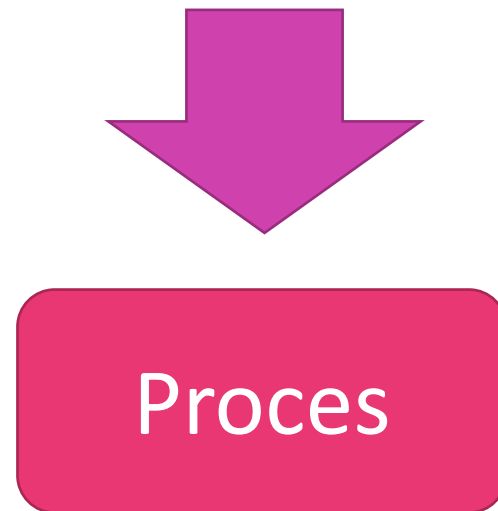


Osnovni pojmovi: Proces

- Proces
 - **Instanca** programa koja se izvodi
 - Jedan program može imati **više** instanci (npr. dva puta se otvori)
 - OS se brine da procesi imaju odvojene resurse (nemoguće je pristupiti varijabli drugog procesa iz prvog procesa)

Kako nastaje proces

- Program u binarnom obliku je učitani u radnu memoriju
- Program rezervira dodatnu memoriju za svoje varijable
- Program rezervira razne resurse operativnog sustava



- Operativni sustav je mozak za alociranje svih potrebnih resursa

Resursi procesa

- Registri (*Register*)
 - Instrukcije, pohrana adresa...
- Brojači (*Counter*)
 - Vodi brigu koja je trenutna i sljedeća instrukcija na izvršavanju
- Stog (*Stack*)
 - služi za pohranu niza istovrsnih elemenata (omogućava upis i ispis po principu "zadnji koji ulazi - prvi izlazi,, - FIFO)
- Dinamički alocirana memorija (*Heap*)



Detaljnije na predmetu Građa računala

Više procesa

- Primjer:
 - Pokrenuto više istih Aplikacija – više procesa i više nezavisnih alokacija memorije
 - Prebacivanje iz jednog procesa u drugi zahtjeva vrijeme
 - Nezavisnost procesa je bitna odlika Operativnog sustava
 - Lakše je pronaći jedan proces koji je „zapeo” nego „gasiti cijelo računalo”

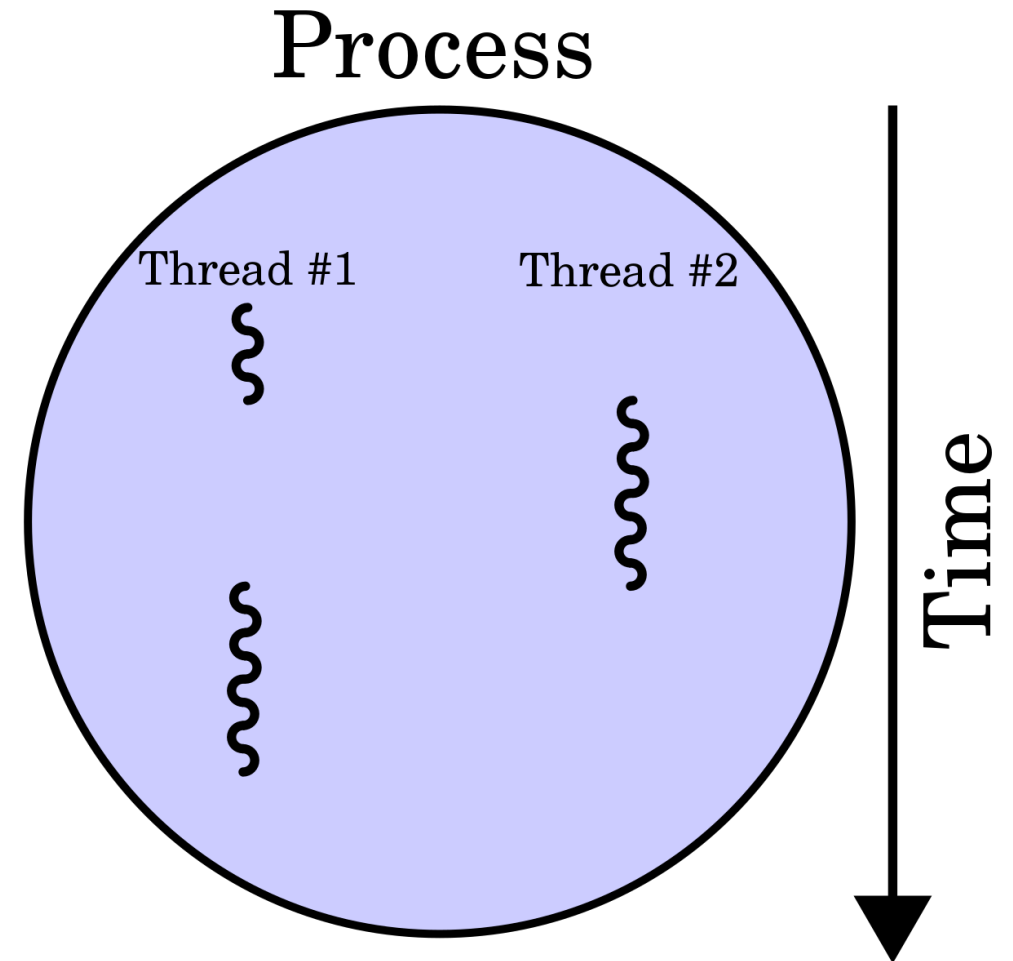
Osnovni pojmovi: Dretva

- Dretva (Thread)
 - Izvršavaju se unutar procesa (kod većine OS-a)
 - Dijelegiraju resurse (Memorija tj. adresni prostor)
 - Istoj varijabli mogu pristupiti različite dretve
 - Imaju zaseban stog (Stack)
 - Na sustavu s jednim procesorom događa se time-division multiplexing
 - Procesorsko vrijeme se prebacuje s jedne na drugu dretvu
 - Korisnik ima dojam da se operacije izvode simultano
 - Na višeprocorskim sustavima dretve se izvode simultano ovisno o broju procesora / jezgara
- Svaki process ima adresni prostor i jednu kontrolnu dretvu

Proces vs. Dretva

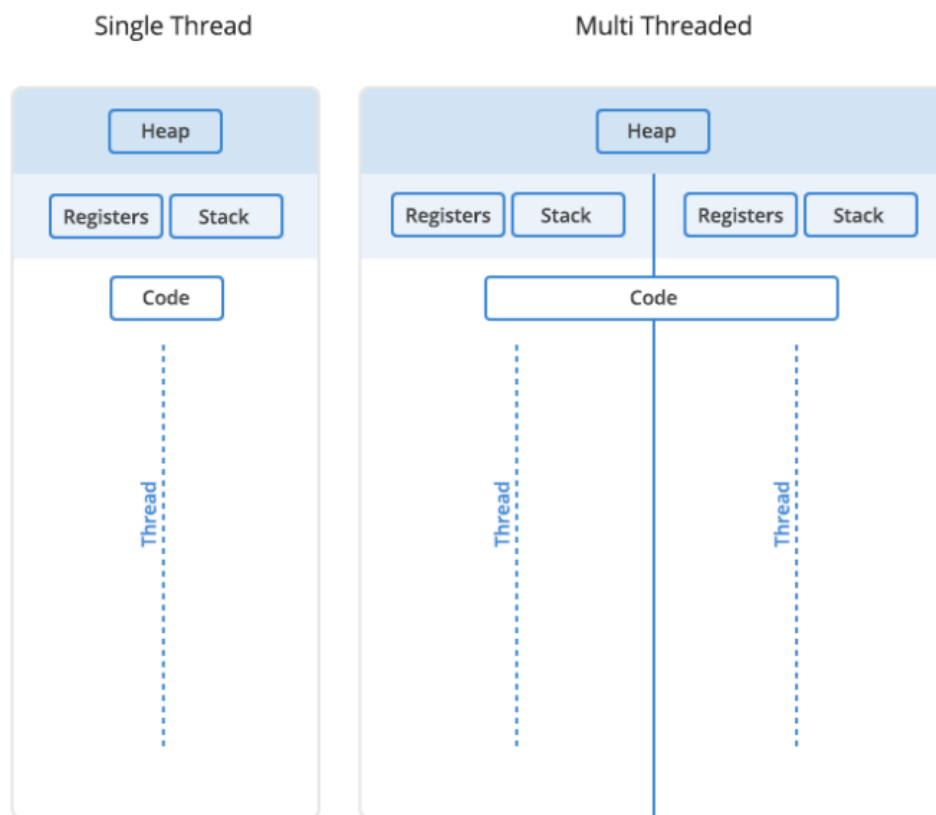
Svojstveno pojedinom procesu	Svojstveno pojedinoj dretvi
Adresni prostor	Programski brojač
Globalne varijable	Registri
Otvorene datoteke	Stog
Procesi djeca	Stanje
Tekući alarmi	
Signali i rukovatelji signalima	

Višedretvenost primjer: jedna dretva čita sa diska, a druga vrši aritmetičke operacije

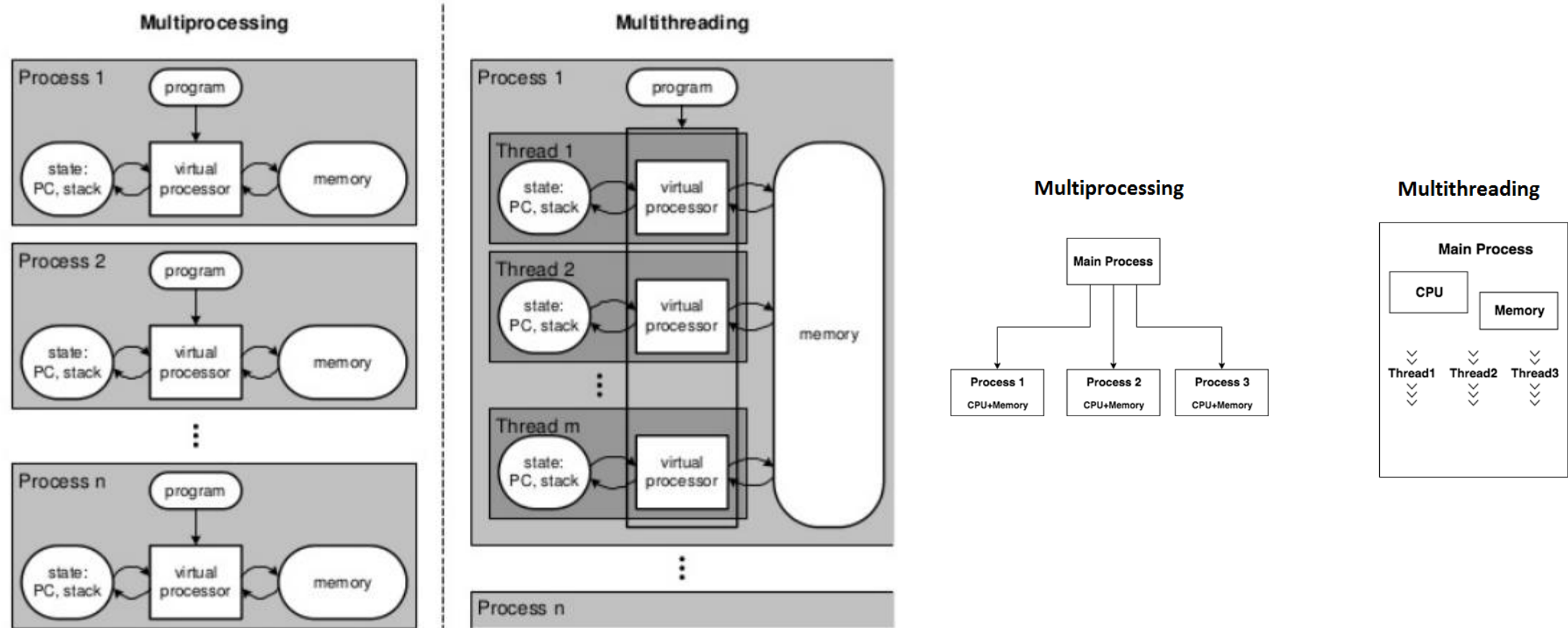


Dretvenost (Thread)

- Jedan proces može imati jednu ili više dretvi



Proces vs. Dretva



Kreiranje procesa

Comparison Factor / Usporedba	<i>fork()</i>	<i>vfork()</i>	<i>exec()</i>	<i>clone()</i>
Invoking/Pozivanje	<i>fork()</i> , creates a child process of the invoking process	<i>vfork()</i> , creates a child process that shares some attributes with the parent	<i>exec()</i> , replaces the invoking process	<i>clone()</i> , creates a child process and offers more control on data shared
Process ID	Parent process and child process have unique Ids	Parent process and child process have the same ID	The process running and the process that replaces it, have the same PID	Parent process and child process have unique IDs but can share when specified
Execution/Izvršavanje	Parent and child processes start simultaneously	Parent process is temporarily suspended while child process runs	Parent process is terminated and the new process starts at entry point	Parent and child processes start simultaneously

Primjer kreiranja procesa i dretvi u C/C++

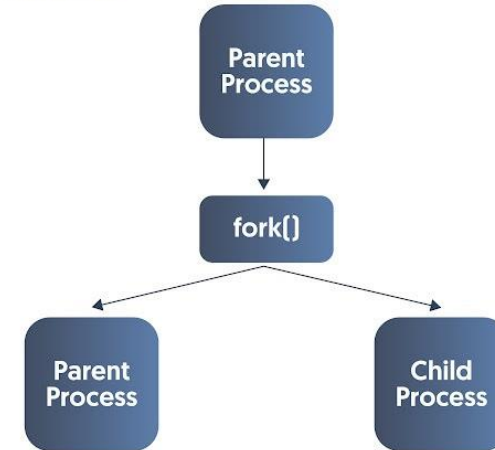
- Funkcije:

- Fork ()
- Clone ()

- Rezultat funkcije `fork`:

- `fork () < 0` – nije uspio proces kreiranja djeteta
- `fork () = 0` – uspješno kreiran proces djeteta
- `fork () > 0` – informacija roditelju. Vrijednost sadrži PID od djeteta

Fork in C



C/C++ primjeri

On-line simulator:

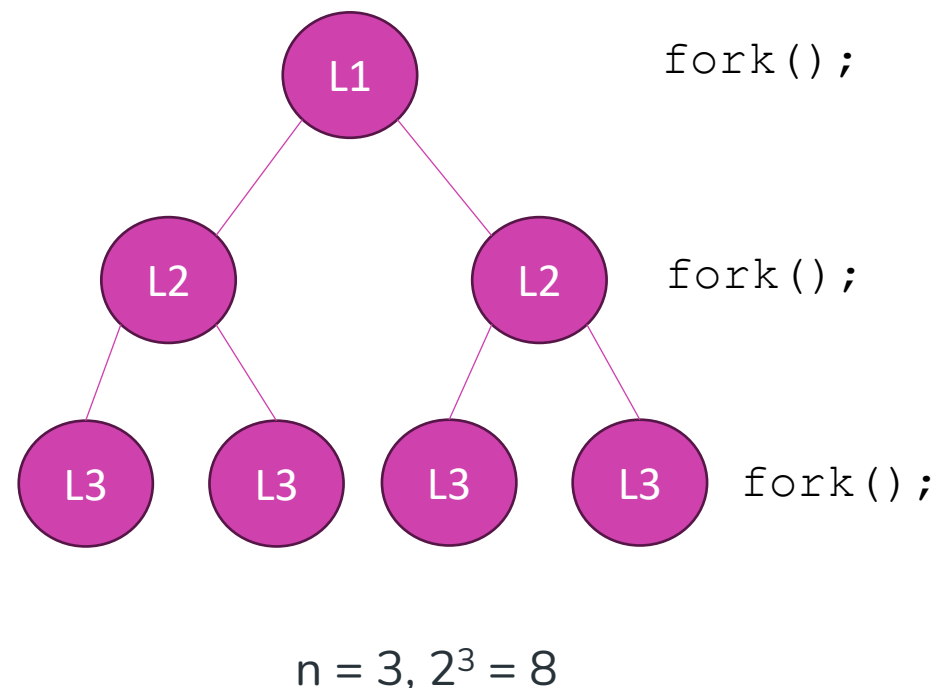
<https://www.onlinegdb.com/>

Fork primjer 1

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main()
{
    fork(); //Linija 1
    fork(); //Linija 2
    fork(); //Linija 3
    printf(„Pozdrav\n“);
    return 0;
}
```

Ispis:

```
Pozdrav
Pozdrav
Pozdrav
Pozdrav
Pozdrav
Pozdrav
Pozdrav
Pozdrav
```



NAPOMENA: Simulaciju možete odraditi na https://www.onlinegdb.com/online_c_compiler

Fork primjer 2

```
#include <stdio.h>
#include <string.h>
#include <sys/types.h>

#define MAX_COUNT 4
#define BUF_SIZE 10000

int main()
{
    pid_t pid;
    int i;
    char buf[BUF_SIZE];
    pid_t p= fork();
    pid = getpid();
    for (i = 1; i <= MAX_COUNT; i++) {
        sprintf(buf, "Ovo je linija od pid %d, sa vrijednoscu = %d, a ja sam=%d\n", pid, i,p);
        write(1, buf, strlen(buf));
    }
}
```

OUTPUT:

Ovo je linija od pid 28076, sa vrijednoscu = 1, a ja sam=28077
Ovo je linija od pid 28076, sa vrijednoscu = 2, a ja sam=28077
Ovo je linija od pid 28076, sa vrijednoscu = 3, a ja sam=28077
Ovo je linija od pid 28076, sa vrijednoscu = 4, a ja sam=28077
Ovo je linija od pid 28077, sa vrijednoscu = 1, a ja sam=0
Ovo je linija od pid 28077, sa vrijednoscu = 2, a ja sam=0
Ovo je linija od pid 28077, sa vrijednoscu = 3, a ja sam=0
Ovo je linija od pid 28077, sa vrijednoscu = 4, a ja sam=0

Fork Primjer 3

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>

#define MAX_COUNT 20

void main(void)
{
    pid_t pid, rez;
    int i;

    for (i=1; i<=MAX_COUNT; i++){
        rez=fork();
        pid = getpid();
        if(rez==-1){
            printf ("Rez=%d A- Ovo je proces (PID) %d kojeg nisam uspio kreirati \n", rez, pid);
        } else if (rez==0){
            printf("Rez=%d B- Ovo je proces Djete/CHILD (PID)=%d, i=%d \n", rez, pid, i);
        } else{
            printf("Rez=%d C- Ovo je proces TATA/Parent (PID)=%d, i=%d \n", rez, pid, i);
            //exit(0);
        }
    }
}
```

OUTPUT:

```
...
Rez=2363 C- Ovo je proces TATA/Parent (PID)=2359, i=1
Rez=0 B- Ovo je proces Djete/CHILD (PID)=2363, i=1
Rez=2364 C- Ovo je proces TATA/Parent (PID)=2359, i=2
Rez=0 B- Ovo je proces Djete/CHILD (PID)=2364, i=2
Rez=2366 C- Ovo je proces TATA/Parent (PID)=2359, i=3
Rez=0 B- Ovo je proces Djete/CHILD (PID)=2366, i=3
...
Rez=2447 C- Ovo je proces TATA/Parent (PID)=2386, i=19
Rez=2446 C- Ovo je proces TATA/Parent (PID)=2396, i=18
Rez=-1 A- Ovo je proces (PID) 2407 kojeg nisam uspio
kreirati
Rez=-1 A- Ovo je proces (PID) 2380 kojeg nisam uspio
kreirati
...
```

Python primjer

Primjer Python:

```
# Python code to create child process
import os

def parent_child():
    n = os.fork()

    # n greater than 0 means parent process
    if n > 0:
        print("Parent process and id is : ", os.getpid())

    # n equals to 0 means child process
    else:
        print("Child process and id is : ", os.getpid())

# Driver code
parent_child()
```

**Primjer kreiranja
dretve**

Jednostavni primjer kreiranja dretve:

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

void* func(void* arg)
{
    // detach the current thread
    // from the calling thread
    pthread_detach(pthread_self());

    printf("Inside the thread\n");

    // exit the current thread
    pthread_exit(NULL);
}

void fun()
{
    pthread_t ptid;
    // Creating a new thread
    pthread_create(&ptid, NULL, &func, NULL);
    printf("This line may be printed"
           " before thread terminates\n");

    // The following line terminates the thread manually
    pthread_cancel(ptid);

    // Compare the two threads created
    if(pthread_equal(ptid, pthread_self()))
        printf("Threads are equal\n");
    else
        printf("Threads are not equal\n");

    // Waiting for the created thread to terminate
    pthread_join(ptid, NULL);

    printf("This line will be printed"
           " after thread ends\n");

    pthread_exit(NULL);
}

// Main code
int main()
{
    fun();
    return 0;
}
```

OUTPUT:

```
This line may be printed before thread terminates
Threads are not equal
Inside the thread
This line will be printed after thread ends
```




**Hvala na
pažnji!**