

Operativni sistemi i konkurentno programiranje

Niti

Koncept konkurentnosti

- Nastao je kao mehanizam za optimizaciju upotrebe procesora. Izuzetno je aktuelan zato što omogućava:
 - **Konukrentno izvršavanje nezavisnih delova programa.** Primer: konukrentno izvršavanje glavnog programa i pozadinskih funkcija.
 - **Bolje i ravnomernije iskorišćenje procesora.** Primer: umesto da procesor čeka da neki proces dobije podatke sa sporog ulaza (konzole), može da izvršava neki drugi proces koji ima podatke.
 - **Potencijalno ubrzanje izvršavanje programa korišćenjem više jezgara/procesora (paralelizacija).** Primer: umesto da jedno jezgro sabira milion elemenata, 4 jezgra mogu da sabiraju po 250000 elemenata u isto vreme.

Konkurentno programiranje

- Može da se realizuje korišćenjem različitih mehanizama. Na ovim vežbama ćemo govoriti o konkurentnom programiranju upotrebom:
 - Niti
 - Kritičnih sekcija

Niti (*threads*)

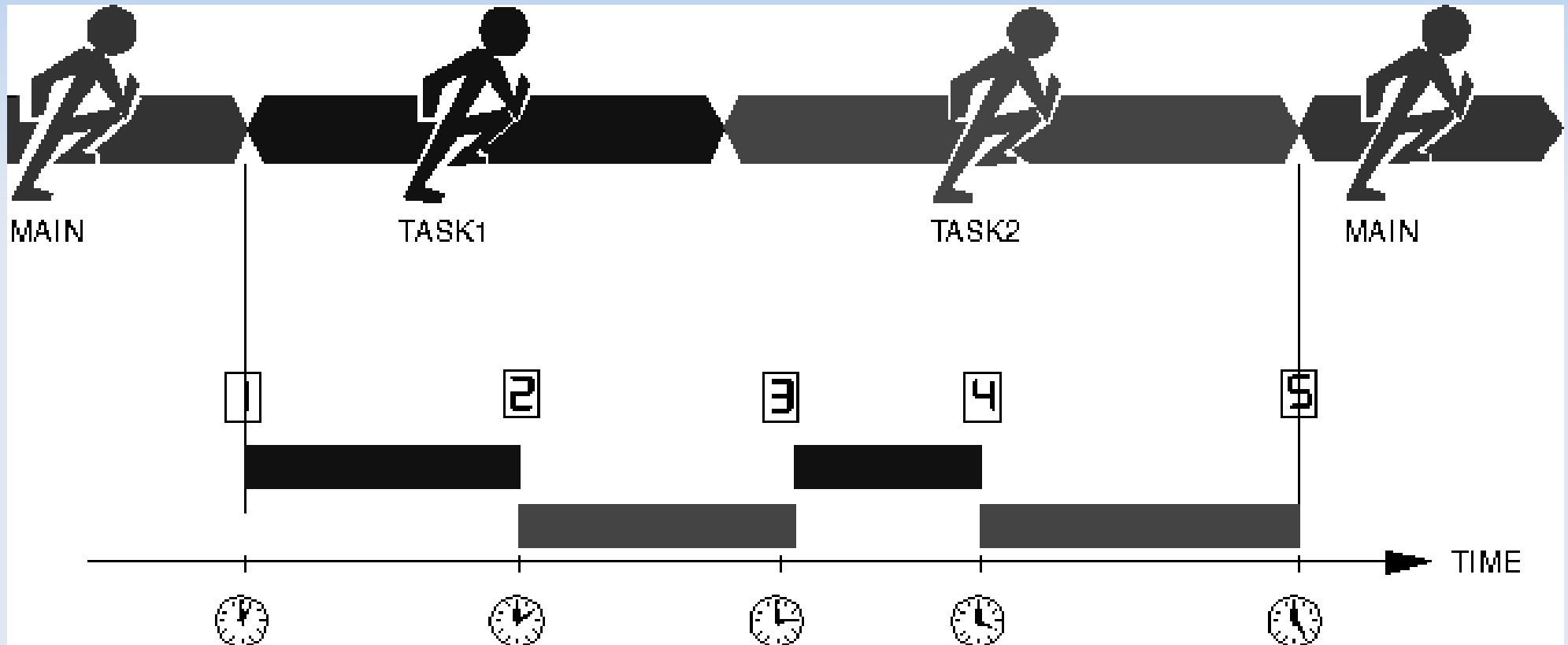
U operativnom sistemu niti predstavljaju:

- **tokove izvršavanja, tj. funkcije programskog koda** (koji se prepliću na procesoru)
- čine osnovu raspoređivanja (*scheduling entity*).

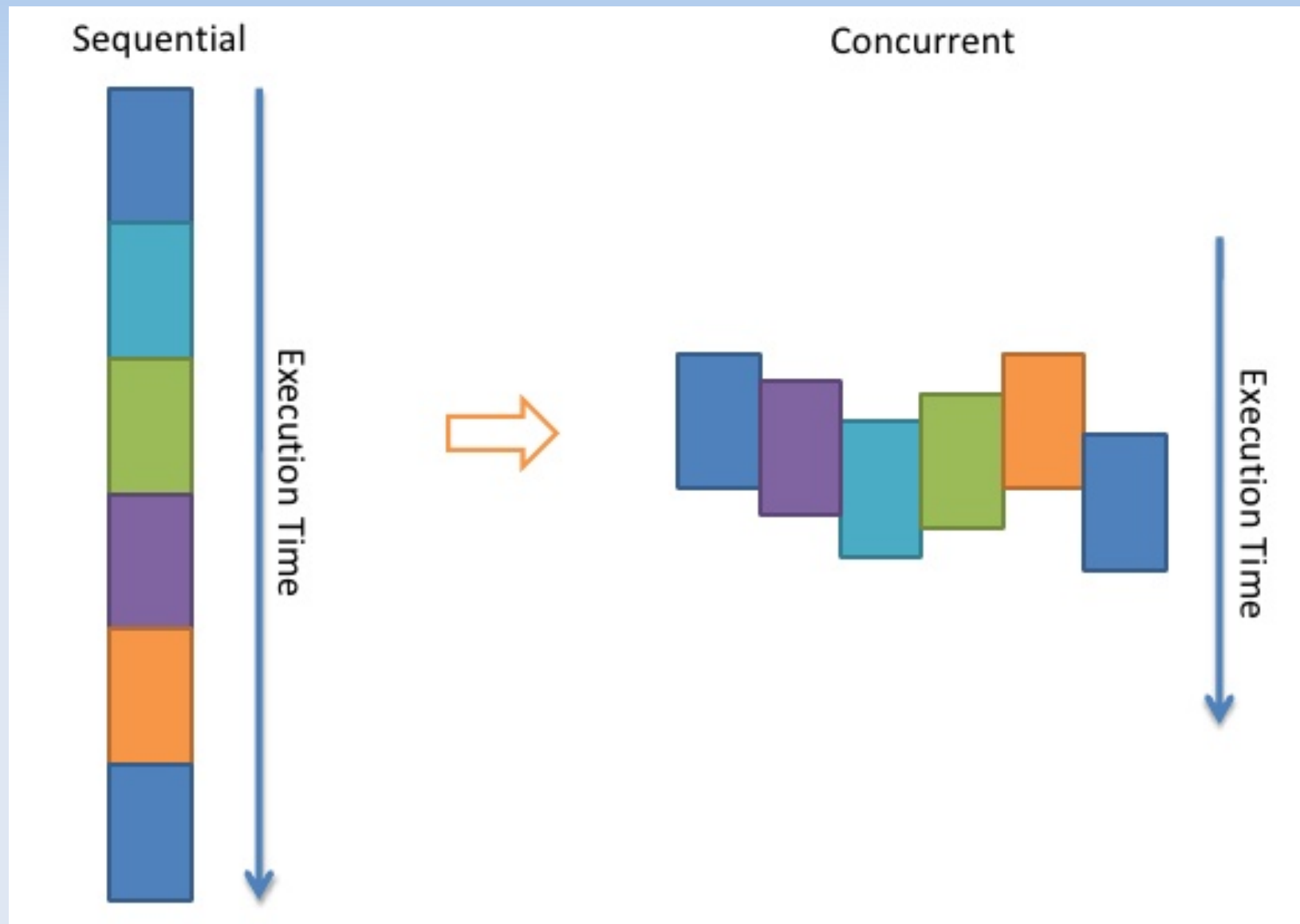
Sekvencijalno ili konkurentno izvršavanje

- Svaki program ima **bar jednu** nit (tok izvršavanja) nastalu od funkcije **main ()**
- Program koji se sastoji od **samo jedne** niti se naziva **sekvencijalni** program
- Program koji se sastoji od **više** (od jedne) niti se naziva **konkurentni** program

Sekvencijalni i konkurentni program u slučaju jednoprocorskog sistema



Sekvencijalni vs konkurentni program u slučaju multiprocesorskog sistema



Od čega nastaju niti?

- Niti nastaju od funkcija
- Svaka nit po svom kreiranju počinje da izvršava funkciju koja je prosleđena konstruktoru niti

Funkcija main

- Ulazna tačka (*entry point*) programa tj. početak korisničkog dela programa je funkcija **main ()**
- Završetak korisničkog dela programa nastupa kada se završi funkcija **main ()**
- Ovo **važi** kako za sekvencijalni tako i za konkurentni program! iz čega sledi...

Nit nastala od funkcije `main()`

je od posebnog značaja jer:

- program traje koliko traje nit `main()`
- kada se završi nit `main()` završava se ceo program (bez obzira na stanja ostalih niti)
- se ni po čemu drugom ne razlikuje od ostalih niti (npr. veći prioritet)

Objekat klase thread (nit)

- Služi za stvaranje niti (toka izvršavanja)
- Kada se stvori nit **objekat** klase thread je u stanju 'joinable' (pokazuje na tok izvršavanja). Ovo znači da je nit krenula da se izvršava.
- se prevodi iz stanja 'joinable' metodama:
 - **join()**
 - **detach()**.
- Ukoliko se nit ne prevede iz stanja 'joinable' nekom od dve metode (**join** ili **detach**) dobiće se greška: "***terminate called without an active exception***".

Razlika između `join()` i `detach()`

- Metoda **`join()`** **blokira** nit pozivaoca dok se nit na kojoj je metoda **`join()`** pozvana **ne završi**
 - *Koristi se kada nit main čeka rezultat rada niti koje je stvorio*
- Metoda **`detach()`** **razdvaja** nit pozivaoca od niti na kojoj je operacija **`detach()`** pozvana, tako da nit pozivaoc **ne čeka** da se nit na kojoj je operacija **`detach()`** pozvana **završi**
 - *Koristi se kada postoje ciklične niti (daemon) koje izvršavaju neku funkciju u beskonačnoj petlji. Nit main() tada uz pomoć `detach()` može da završi rad, a u suprotnom pozivom metode `join()` bi čekala beskonačno.*

Argumenti i povratna vrednost

- Ako je nit nastala od funkcije $f()$:
 - Pri stvaranju, niti se moraju proslediti argumenti po istim pravilima kao da se poziva obična C funkcija $f()$
 - Vrednosti svih argumenata niti se pri stvaranju kopiraju u kontekst niti
 - Povratna vrednost funkcije $f()$ se zanemaruje (uvek je **void**)