Date: ..../..../.....

id = ∅

② خیر، زیرا اگر process با ~~id=1~~ وارد ناحیه بحرانی شود

(برای بار اول) در آن صورت process با id=1 در حلقه

while(blocked[1-id]) گیر کرده است. حال فرض کنید process با id=∅

به blocked[id] = false برسد. سپس با سرعت به بالای حلقه رفته و

blocked[id] = true خواهد کرد و به همین ترتیب وارد critical section شود. سپس

CPU کنترل را به process با id=1 واگذار کند. آنگاه این process

از حلقه while(blocked[1-id]) بیرون آمده و turn=id=1 می شود

سپس به حلقه while(turn!=id) می رسد و چون شرط برقرار

نیست وارد حلقه نمی شود پس وارد critical section

می شود. حال دو process در critical section حضور

دارند، پس mutual exclusion نقض می شود.

Date: / /

③ سناریو: وقتی خط printf(ping) اجرا شده سراغ mini_thread_start می‌رویم. در این لحظه thread بعدی ساخته می‌شود. حال CPU می‌تواند به سراغ اجرای thread ساخته شده (pongthread) برود یا thread فعلی (pingthread) را اجرا کند. اگر thread فعلی را اجرا کند سراغ mini_thread_stop رفته و pongthread را متوقف کرده و مجددا ping is here را چاپ می‌کند که خلاف میل ماست.

```
void ping() {
    while(true) {
        printf("ping")
        signal(s)
        wait(u)
    }
}
```

```
void pong() {
    while(true) {
        wait(s)
        printf("pong")
        signal(u)
    }
}
```

مقدار اولیه سمافورها برابر صفر است.

s, u