Линда

Библиотека C-Linda

- **out**(t) – ставља нову пасивну торку t у простор торки, након што се изврши израчунавање поља торке; позивајући процес одмах наставља рад, не блокирајући се.

Библиотека C-Linda

- **in**(p) – тражи пасивну торку t у простору торки која одговара торки-шаблону p (торка-шаблон је торка чија су поља или вредности или променљиве којима претходи знак питања). Ако је не нађе, процес позивалац се суспендује (блокира) док се одговарајућа торка не појави; ако је нађе, додељује вредности поља торке t одговарајућим променљивама из торке-шаблона p, а потом брише нађену торку из простора торки. Ако постоји више торки које одговарају торки-шаблону p, недетерминистички се бира једна од њих.

Библиотека C-Linda

- **rd**(p) – има исто понашање као in(p), само што се пронађена торка не брише из простора торки.

Библиотека C-Linda

- **eval**(t) – ставља активну торку t у простор торки; у суштини има исто понашање као out(t), сам што се израчунавање поља торке ради након стављања у простор торки. За свако поље торке t које садржи функцију која враћа неку вредност, имплицитно може да се креира нов процес. Позивајући процес одмах наставља рад, не блокирајући се. Када сва активна поља буду израчуната (процес(и) креирани над функцијама заврше рад и врате вредност), торка постаје пасивна.

Библиотека C-Linda

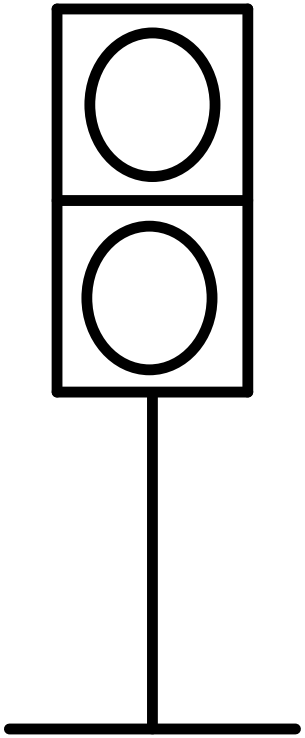
- **inp**(p) - тражи пасивну торку t у простору торки која одговара торки-шаблону p. Ако је не нађе враћа FALSE; ако је нађе, додељује вредности поља торке t променљивама из торке-шаблона p, брише торку t из простора торки и враћа TRUE. Ако постоји више торки које одговарају торки-шаблону p, недетерминистички се бира једна од њих.

Библиотека C-Linda

- **rdp**(p) – има исто понашање као inp(p), само што се пронађена торка не брише из простора торки.

Задаци

Семафор



Семафор

Користећи библиотеку C-Linda приказати како се могу креирати “Семафори”.

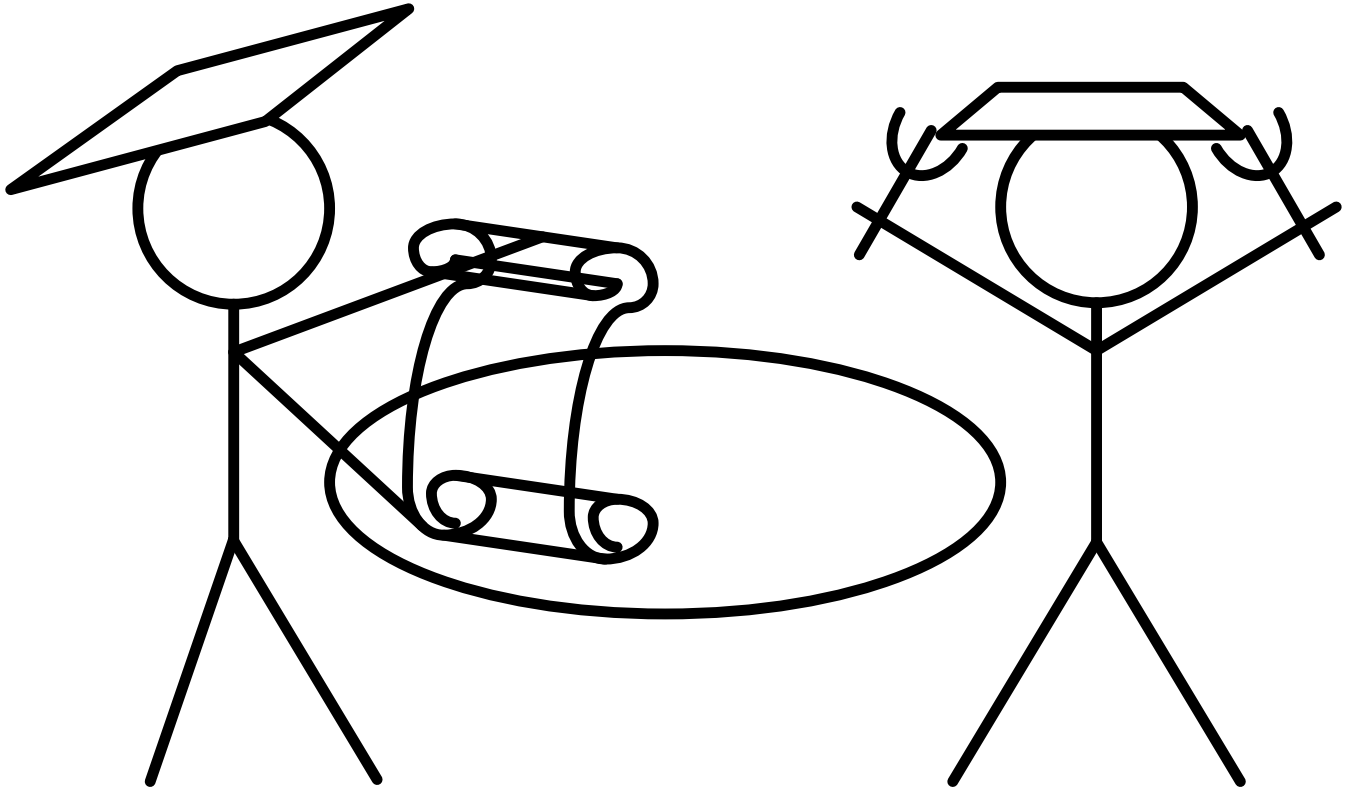
Семафор

signal: **out** (“sem”)
wait: **in** (“sem”)
Иницијализација на n тако што се **out** (“sem”) изврши n пута.

Или у виду метода:

```
void signal(String sem){
    out(sem);
}
void wait(String sem){
    in(sem);
}
void init(String sem, unsigned int val){
    for(int i = 0; i < val; i++){
        out(sem);
    }
}
```

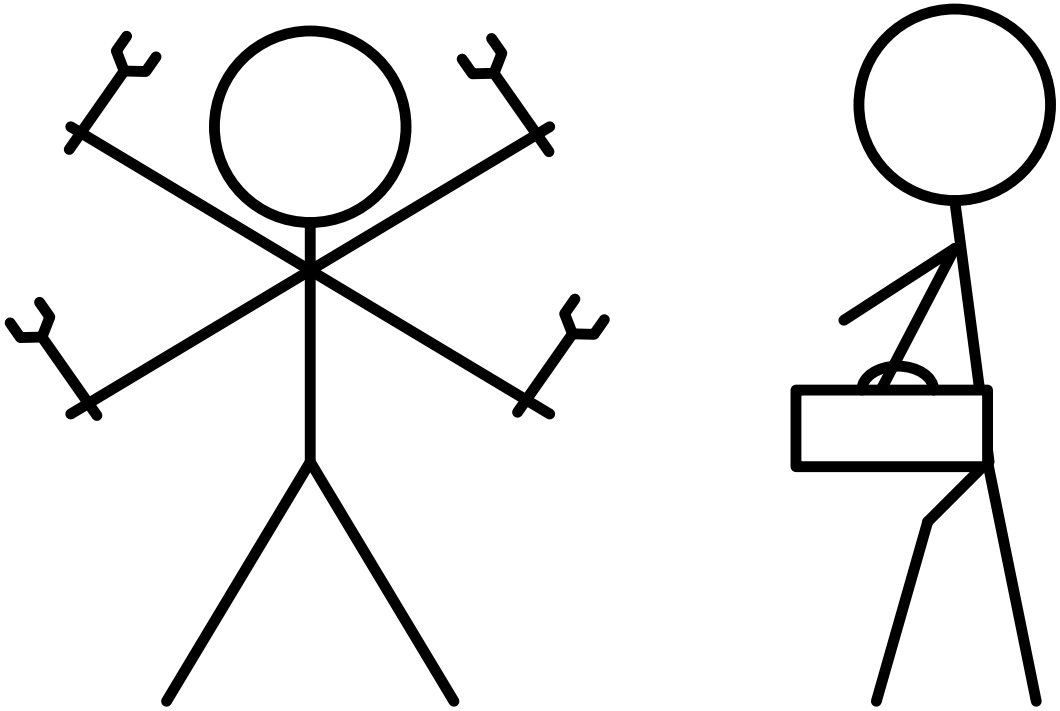
Dining philosophers problem



Dining philosophers problem

```
void phil (int i) {
    int i;
    while (1) {
        think ();
        in (“room ticket”);
        in (“chopstick”, i);
        in (“chopstick”, (i+1)%Num);
        eat ();
        out (“chopstick”, i);
        out (“chopstick”, (i+1)%Num);
        out (“room ticket”);
    }
}
void initialize () {
    int i;
    for (i=0; i<Num; i++) {
        out (“chopstick”, i);
        eval (phil (i));
        if (i<(Num-1)) out (“room ticket”);
    }
}
```

Клијент-сервер



Клијент-сервер

У једном дистрибуираном рачунарском систему има више клијената и један сервер. Број клијената је произвољан. Клијент шаље захтев, а сервер га опслужује и шаље одговор клијенту. Захтев који је раније послат има предност над оним који је послат касније.

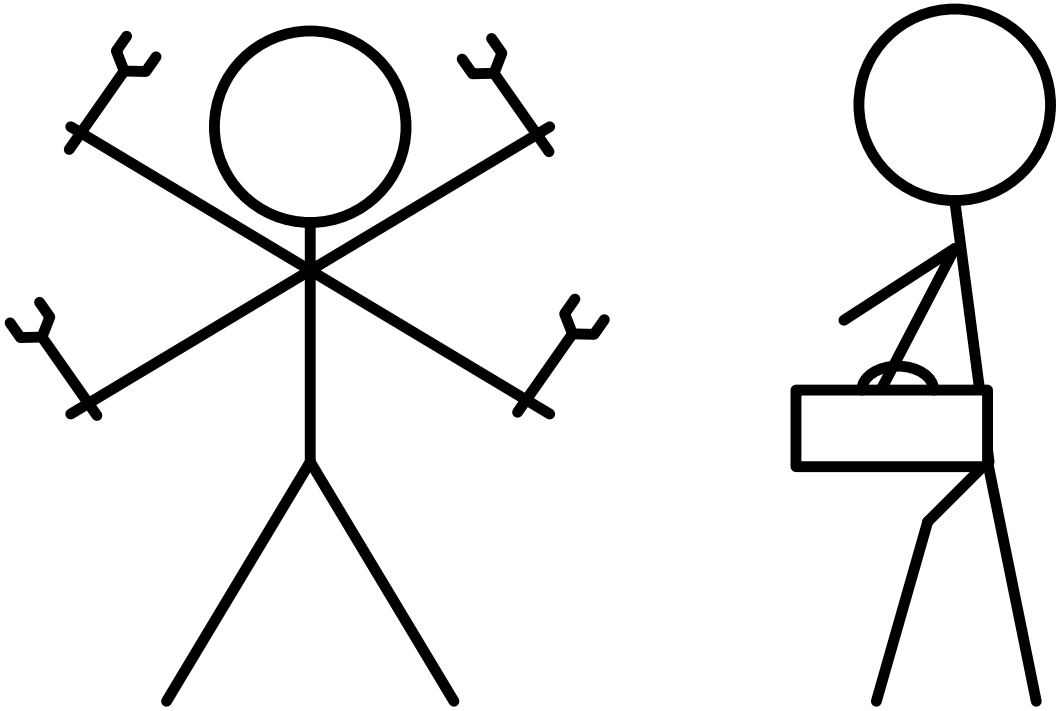
Клијент-сервер

```
void server () {
    int index = 1;
    ...
    while (1) {
        in (“request”, index, ?req);
        ...
        out (“response”, index++, response);
    }
}
void client () {
    int index;
    ...
    in (“server index”, ?index);
    out (“server index”, index+1);
    ...
    out (“request”, index, request);
    in (“response”, index, ?response);
    ...
}
```

Клијент-сервер

```
void inicijalizacija () {
      out ("server index ", 1);
      eval (server());
      for (int i=0; i<10; i++)
            eval (client());
}
```

Клијент-сервер



Клијент-сервер

У једном дистрибуираном рачунарском систему има више клијената и више сервера. Број клијената је произвољан, а број сервера се налази у простору торки. Клијент шаље захтев, а један од сервера (било који) га опслужује и шаље одговор клијенту. Захтев који је раније послат има предност над оним који је послат касније. Постоји и процес *sumator* који с времена на време захтева од свих сервера да му пошаљу број обрађених захтева до тог тренутка и то тако што сви сервери заврше са обрадом захтева на коме тренутно раде, шаљу том процесу тражени податак и не узимају нови захтев све док овај процес не добије податке од свих сервера. Написати функције *klijent*, *server* и *sumator* који реализују наведене операције и код за иницијализацију система (ако је потребан) користећи библиотеку C-Linda.

Клијент-сервер

У prostoru torki (tuple space) се могу наћи следеће торке:

- ("number of servers", **int** n) - укупан број сервера
- ("summing") - у простору торки када се врши сумирање
- ("not summing“)- у простору торки када се не врши сумирање
- ("client index", **int** i) - редни број следећег клијентског захтева ("server index", **int** i) - редни број следећег серверског захтева
- ("request", **int** redni_broj, **int** param) - захтев са одговарајућим редним бројем
- ("response", **int** redni_broj, **int** rezultat) - одговор на захтев са одговарајућим редним бројем
- ("number of processed", **int** broj_obradenih) - одговор на упит о броју обрађених захтева

Клијент-сервер

```
void client (int id) {
    int index;
    ...
    in (“client index”, ?index);
    out (“client index”, index+1);
    ...
    out (“request”, index, request);
    in (“response”, index, ?response);
    ...
}
```

Клијент-сервер

```
void server () {
    int i=1, cnt=0, x;
    in ("number of servers", ?x);
    out ("number of servers", x + 1);
    while (1)
        if (rdp ("summing")) {
            out ("number of processed", cnt);
            rd ("not summing");
        }
        else    {
            in ("server index", ?i);
            out ("server index", ++i);
            in ("request", i, ?x);
            out ("response", i, f(x));
            cnt++;
        }
}
```

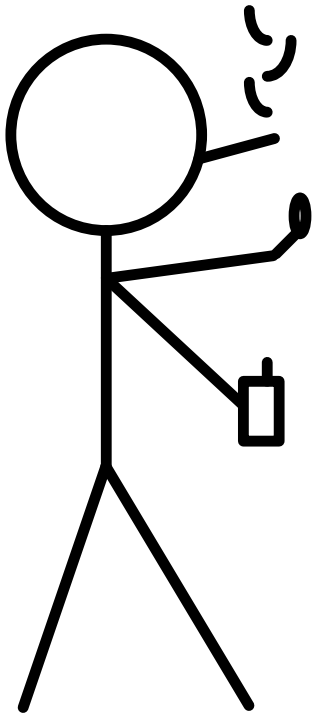
Клијент-сервер

```
void sumator () {
    int n, ukupno, j;
    ....
    in (“not summing”);
    out ("summing");
    rd ("number of servers", ?n);
    cnt = 0;
    for (int i = 0; i < n; i++) {
        in ("number of processed", ?j);
        cnt += j;
    };
    in (“summing");
    out (“not summing”)
    ...
}
```

Клијент-сервер

```
void init () {
      out ("number of servers", 0);
      out ("client index", 0);
      out ("server index", 0);
      out (“not summing”);
      for (int i = 0; i < 10; i++)
            eval (server ());
}
```

Cigarette Smokers' problem



Cigarette Smokers' problem

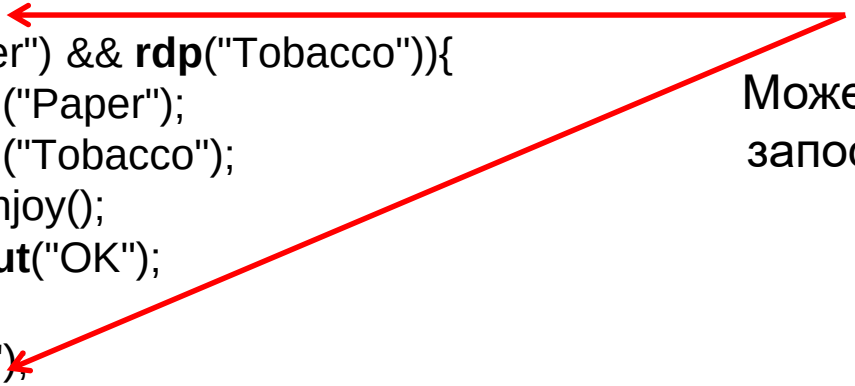
Користећи C-Lindu написати програм који решава проблем и симулира систем "нервозних пушача". Постоји један агент и три нервозна пушача. Агент поседује резерве три неопходна предмета за лечење нервозе: папир, дуван и шибице. Један од пушача има бесконачне залихе папира, други – дувана, а трећи - шибица. Агент почиње тако што два различита предмета ставља на сто, један по један. Пушач, коме баш та два предмета фале, узима их, завија и пали цигарету и ужива. Након тога обавештава агента да је завршио, а агент онда ставља два нова предмета на сто, итд.

Cigarette Smokers' problem

```
void agent(){
    int n;
    while(1){
        n = (int)((rand()*3)/RAND_MAX);
        switch(n){
            case 0:  out("Paper");
                     out("Tobacco");
                     break;
            case 1:  out("Tobacco");
                     out("Matches");
                     break;
            case 2:  out("Matches");
                     out("Paper");
                     break;
        }
        in("OK");
    }
}
```

Cigarette Smokers' problem

```
void smocker_with_matches(){
    while(1){
        in("Watch");
        if(rdp("Paper") && rdp("Tobacco")){
            in("Paper");
            in("Tobacco");
            enjoy();
            out("OK");
        }
        out("Watch");
    }
}
```

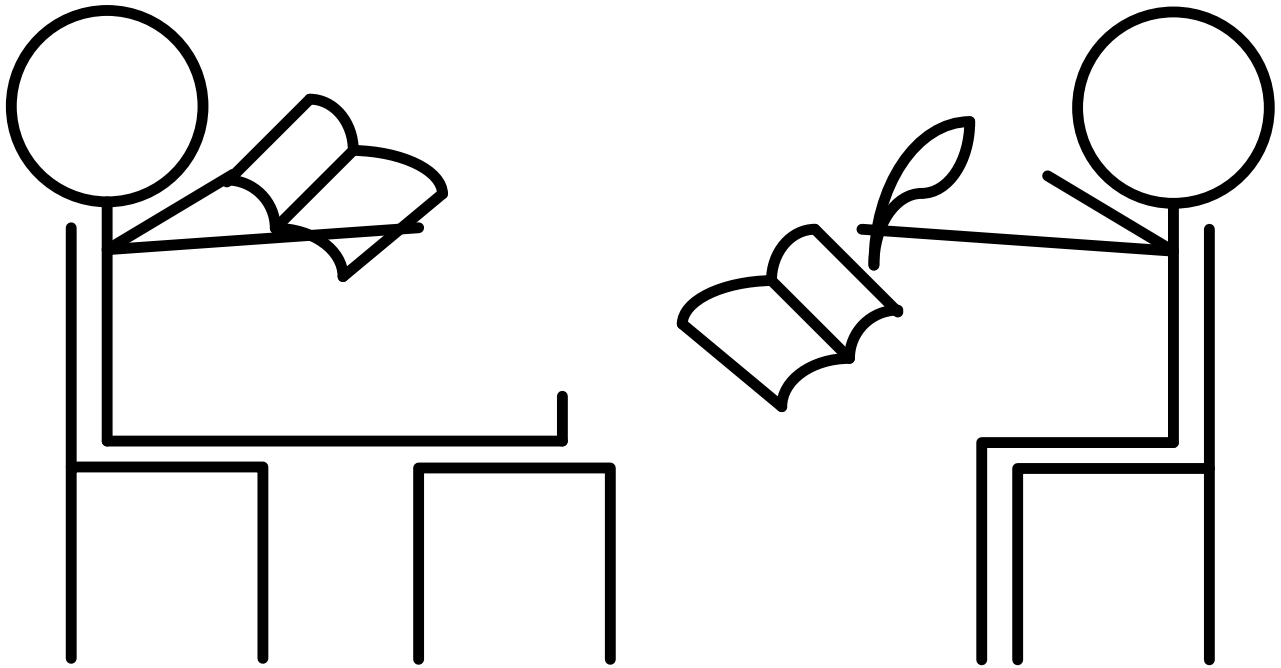


Може да доведе до запосленог чекања

Cigarette Smokers' problem

```
initialize () {
      eval(agent());
      eval(smocker_with_matches ());
      eval(smocker_with_paper());
      eval(smocker_with_tobacco());
      out("Watch");
}
```

Readers – Writers problem



Readers – Writers problem

Користећи C Lindu решити проблем читалаца и писаца. Решење треба да обезбеди да процес који је пре стигао пре и започне операцију читања односно уписа.

Readers – Writers problem

```
void reader(){
    int id, num;
    while(1){
        in("id", ?id);
        out("id", id + 1);
        in("ok_to_work", id);
        in("readers_num", ?num);
        out("readers_num", num + 1);
        out("ok_to_work", id + 1);
        reading();
        in("readers_num", ?num);
        out("readers_num", num - 1);
    }
}
```

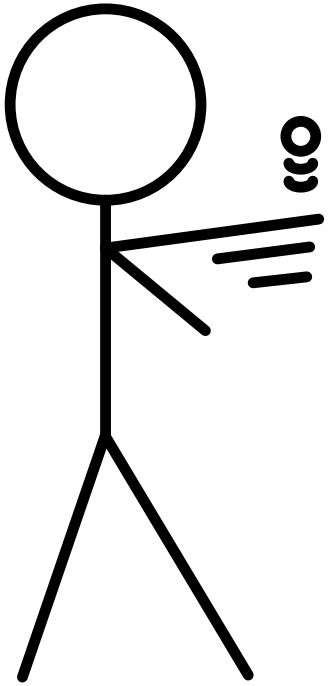
Readers – Writers problem

```
void writer(){
    int id;
    while(1){
        in("id", ?id);
        out("id", id + 1);
        in("ok_to_work", id);
        rd("readers_num", 0);
        writing();
        out("ok_to_work", id + 1);
    }
}
```

Readers – Writers problem

```
void init () {
        int i;
        out("id", 0);
        out("ok_to_work", 0);
        out("readers_num", 0);
        for (i=0; i<10; i++) {
                eval (reader ());
                eval (writer ());
        }
}
```

Проблем избора



Проблем избора

Користећи C-Lindu написати програм који решава следећи проблем: Постоје три особе међу којима треба изабрати једну. Свака од тих особа поседује новчић који има две стране. Избор особе се одиграва тако што свака особа независно баца свој новчић. Уколико постоји особа којој је новчић пао на другу страну у односу на преостале две онда се та особа изабира. Уколико све три особе имају исто постављен новчић поступак се понавља све док се не изабере једна.

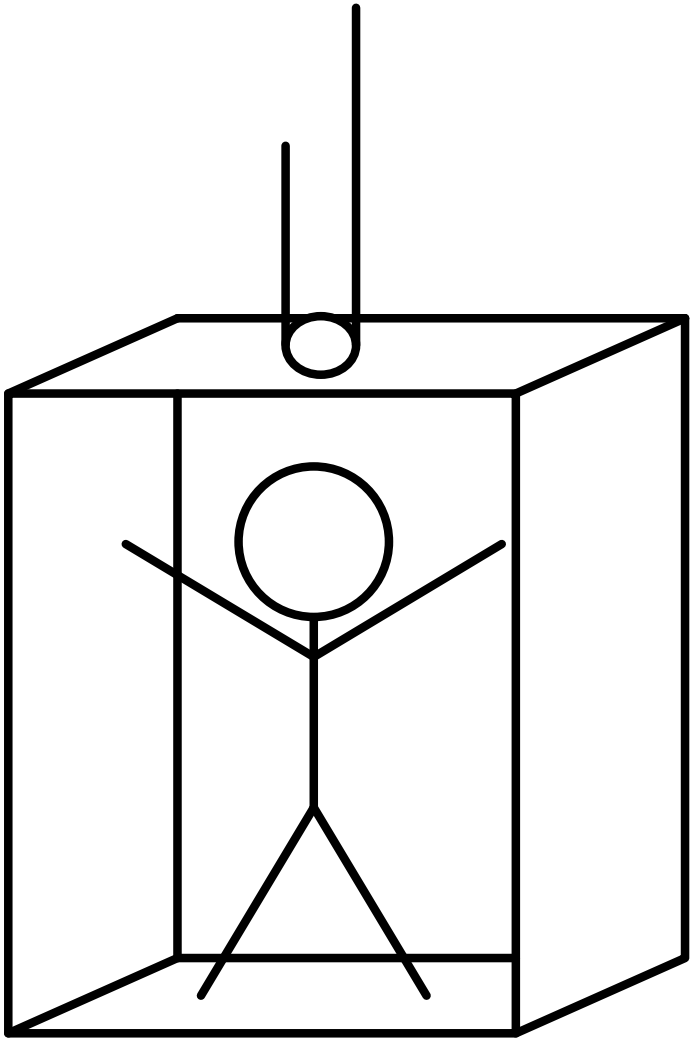
Проблем избора

```
void Player(int id){
    int PID;
    int coin, coinl, coinr;
    int end, winer;
    in("PID", ?PID);
    do{
        PID ++;
        coin = (int)((rand()*2)/RAND_MAX);
        out("RESULT", id, PID, coin);
        out("RESULT", id, PID, coin);
        in("RESULT", (id + 1)%3, PID, ?coinr);
        in("RESULT", (id + 2)%3, PID, ?coinl);
        end = !((coin == coinl) && (coin == coinr) && (coinl == coinr));
        winner = (coin != coinl) && (coin != coinr);
    }while(!end);
}
```

Проблем избора

```
void init () {
      out("PID", 0);
      out("PID", 0);
      out("PID", 0);
      eval (Player(0));
      eval (Player(1));
      eval (Player(2));
}
```

Проблем лифтова



Проблем лифтова

Користећи C-Lindu написати програм који решава проблем путовања лифтом. Путник позива лифт са произвољног спрата. Када лифт стигне на неки спрат сви путници који су изразили жељу да сиђу на том спрату обавезно изађу. Након изласка путника сви путници који су чекали на улазак уђу у лифт и кажу на који спрат желе да пређу. Тек када се сви изјасне лифт прелази даље. Није потребно оптимизовати пут лифта и путника.

Проблем лифтова

```
void elevator(){
    int x;
    while(1){
        x = getFloor();

        out("off", x);
        in("stop", x, 0);
        in("off", x);
        out("stop", x, 0);

        out("on", x);
        in("start", x, 0);
        in("on", x);
        out("start", x, 0);
    }
}
```

Проблем лифтова

```
int getFloor(){
    int x;
    in("floor", ?x);
    return x;
}
```

Проблем лифтова

```
void passenger(unsigned ID, x, y){
    int s1, s2;
    if(x != y){
        in("start", x, ?s1);
        if(s1 == 0) out("floor", x);
        out("start", x, s1+1);

        in("on", x);
        in("start", x, ?s1);
        in("stop", y, ?s2);
        if(s2 == 0) out("floor", y);
        inp("floor", x);
```

Проблем лифтова

```
            out("stop", y, s2+1);
            out("start", x, s1-1);
            out("on", x);

            in("off", y);
            in("stop", y, ?s2);
            inp("floor", y);
            out("stop", y, s2-1);
            out("off", y);
    }
}
```

Проблем лифтова

```
void init(){
    int floorNum = N;
    int i;
    for(i=0; i < floorNum; i++){
        out("start", i, 0);
        out("stop", i, 0);
    }
    eval( elevator());
}
```

Питања?

Захарије Радивојевић
Електротехнички Факултет
Универзитет у Београду
zaki@etf.rs