// 2019 Jun K2.

// 1.

class Skola {

condition odeljenja[8][9];

condition sledeci;

int cnt[8][9];

void udji(int razred, int odeljenje) {

cnt[razred][odeljenje]++;

odeljenja[razred][odeljenje].wait();

cnt[razred][odeljenje]--;

if (cnt[razred[odeljenje]]==0)

sledeci.signal();

}

void krenite\_da\_ulazite() {

for (int r = 0; r < 8; r++) {

for (int o = 0; o < 9; o++) {

if (broj[r][o] > 0) {

odeljenja[r][o].signal\_all();

sledeci.wait();

}

}

}

}

};

/\* lose jer kad se uradi signal\_all ne mora da znaci da su svi usli

class School {

condition classes[8][9];

void lets\_enter() {

for (int g = 0; g < 8; g++) {

for (int c = 0; c < 9; c++) {

classes[g][c].signal\_all();

}

}

}

void i\_want\_in\_school(int grade, int class) {

classes[grade][class].wait();

// nema uslov jer su svi u dvoristu pre lets\_enter()

}

}; \*/

// 2.

struct RW {

int nr = 0;

int turn = 0, next = 0;

};

shared RW rw;

void start\_read() {

region(rw) {

int myTurn = turn++;

await(myTurn == next);

next++;

nr += 1;

}

}

void stop\_read() {

region(rw) {

nr -= 1;

}

}

void start\_write() {

region(rw) {

int myTurn = turn++;

await(myTurn == next && nr == 0);

}

}

void stop\_write() {

region(rw) {

next++;

}

}

// 2018 Jun

// 1.

// SC disciplina

monitor Cashier {

condition cashiers[MAX\_N];

int turn[MAX\_N];

int number\_in\_queue[MAX\_N];

bool active[MAX\_N];

condition waitForWork;

condition waitNewCustomers[MAX\_N];

int cashiersToOpen = 0;

int choose\_cashier() {

int cashier\_id = get\_shortest();

int my\_turn = turn[cashier\_id]++;

number\_in\_queue[cashier\_id] += 1;

if (number\_in\_queue[cashier\_id] > LIMIT) {

cashiersToOpen++;

waitForWork.signal();

}

waitNewCustomers[cashier\_id].signal();

cashiers[cashier\_id].wait(my\_turn);

na\_kasi[cashier\_id] = true;

return cashier\_id;

}

void i\_leave() {

na\_kasi[cashier\_id] = false;

finished[cashier\_id].signal();

}

int ready\_for\_work() {

// potrebna nova kasa

while (cashiersToOpen == 0) {

waitForWork.wait();

}

cashiersToOpen--;

}

bool next\_customer(int cashier\_id) {

// izdravavanje racuna

if (na\_kasi[cashier\_id]) finished[cashier\_id].wait();

if (number\_in\_queue[cashier\_id] == 0 && getNumActive() > 1) {

active[cashier\_id] = false;

return false;

}

if (!cashiers[cashier\_id].queue()) {

waitNewCustomers.wait();

}

cashiers[cashier\_id].signal();

number\_in\_queue[cashier\_id] --;

return true;

}

}

process Customer {

while (true) {

int cashier\_id = choose\_cashier();

roba();

plati();

i\_leave(cashier\_id);

}

}

process Worker {

while (true) {

ready\_for\_work();

while (next\_customer()) {

racun\_je();

}

}

}

// 2.

struct H2O {

Molecule molecule;

int consumed\_molecule;

queue<int> hydrogen;

int o\_place = 0;

int h\_turn = 0, h\_next = 0;

int o\_turn = 0, o\_next = 0;

};

shared H2O h2o;

Molecule hydrogen\_interact(int id) {

Molecule my\_molecule;

region (h2o) {

hydrogen.push\_back(id);

int my\_turn = h\_turn++;

await(my\_turn <= h\_next);

my\_molecule = h2o.my\_molecule;

h2o.consumed\_molecule -= 1;

if (h2o.consumed\_molecule == 0) {

h2o.o\_next++;

}

}

return my\_molecule;

}

Molecule oxygen\_interact(int id) {

Molecule my\_molecule;

region (h2o) {

int my\_turn = h2o.o\_turn++;

await(my\_turn == h2o.o\_next);

await(h2o.hydrogen.size() >= 2);

my\_molecule = Molecule(id, h2o.hydrogen.pop(), h2o.hydrogen.pop());

h2o.molecule = my\_molecule;

h2o.consumed\_molecule = 2;

h\_next += 2;

}

return my\_molecule;

}

// 2017 Jun

class Printers {

bool printers\_busy[2];

condition free\_printer;

int request() {

while (printers\_busy[0] && printers\_busy[1]) {

free\_printer.wait();

}

for (int i = 0; i < 2; i++) {

if(!printers\_busy[i]) {

printers\_busy[i] = true;

return i;

}

}

}

void relase(int printer) {

printers\_busy[printer] = false;

free\_printer.signal();

}

};

// 2019 Septembar

// 1. Uvedene pretpostavke:

// roditelji ostavljaju decu po redosledu dolaska

// ostavljanje dece ima veci prioritet od odlaska dadilje

class ChildCareProblem {

int children = 0;

int nanny = 0;

queue<int> parentsChildren;

int parents\_turn = 0;

condition parents, goNanny;

void put\_child(int children\_num) {

if (children + children\_num > nanny \* 3 || parentsChildren.size() > 0) {

int myTurn = parents\_turn++;

parentsChildren.push\_back(children\_num);

parents.wait(myTurn);

}

children += children\_num;

check\_parent\_put();

}

void get\_child(int children\_num) {

children -= children\_num;

check\_parent\_put();

check\_nanny\_go();

}

void work\_nanny() {

nanny += 1;

check\_parent\_put();

check\_nanny\_go();

}

void home\_nanny() {

if (children > (nanny - 1) \* 3) {

goNanny.wait();

}

nanny -= 1;

check\_nanny\_go();

}

private:

void check\_nanny\_go() {

if (children > (nanny - 1) \* 3)

goNanny.signal();

}

void check\_parent\_put() {

if (parentsChildren.size() > 0 &&

(children + parentsChildren.front() <= nanny \* 3)) {

parentsChildren.pop\_front();

parents.signal();

}

}

}

// 2018 Oktobar

class ResourceAllocation {

int free\_resources = 3;

condition cond[2][2];

void allocate(int n, bool priority) {

if (free\_resources < n)

cond[n - 1][priority].wait();

else

free\_resources -= n;

}

void release(int n) {

free\_resources += n;

for (int p = 1; p >= 0; p--) {

for (int c = 1; c >= 0; c--) {

while (cond[c][p].queue() && free\_resources >= c) {

free\_resources -= c;

cond[c][p].signal();

}

}

}

}

}

// 2018 Februar

// POGLEDATI

class BumperCarProblem {

int carTurn = 0;

int customerTurn = 0;

condition customer, car;

void car\_get\_customer() {

if (customer.queue()) {

customer.signal();

} else {

int myTurn = carTurn++;

car.wait(myTurn);

}

}

void car\_free() {

}

void get\_car() {

if (car.queue()) {

car.signal();

} else {

int myTurn = customerTurn++;

customer.wait(myTurn);

}

}

void put\_car() {

}

}

process Car() {

while (true) {

car\_get\_customer();

car\_free();

}

}

process Child() {

while (true) {

int car = get\_car();

ride();

put\_car(car);

}

}

// 2019 K2

// 1.

monitor Philosophers {

bool forks[5];

int turn = 0;

condition c;

void getForks(int i) {

if (c.queue() || !forks[left] || !forks[right]) {

int my\_turn = turn++;

c.wait(my\_turn \* 5 + i);

} else

forks[left] = forks[right] = false;

}

void releaseForks(int i) {

forks[left] = forks[right] = true;

if (check()) check();

}

private:

bool check() {

if (!c.queue()) return false;

int philo = c.minrank() % 5;

if (forks[philo.left] && forks[philo.right]) {

forks[philo.left] = forks[philo.right] = false;

c.signal();

return true;

}

return false;

}

}

// 2. ACA BRACA

struct Customer {

int barber;

bool startHaircut = false;

bool goHome = false;

};

struct BarberShop {

int ticket = 0;

list<int> waiting;

int customers[3] = {0,0,0}

};

shared BarberShop bs;

void get\_haircut(int preference) { // 0-Bilo ko, 1-Aca, 2-Braca

bool canGet;

Customer customer;

region (bs) {

if (bs.waiting.size() == 15) {

canGet = false;

}

else {

bs.customers[preference]++;

customer = new Customer(preference);

bs.waiting.push\_back(customer);

}

}

if (!canGet) return; // nema mesta, aj kuci

region (customer) {

await(customer.startHaircut);

customer.startHaircut = false; // seo u stolicu

await(customer.goHome);

customer.goHome = false; // platio i otisao kuci

}

}

void get\_customer(int barber) {

Customer customer;

region (bs) {

await(bs.customers[barber]>0 || bs.customers[0]>0);

for (auto it = bs.waiting.iterator(); it != bs.waiting.end(); ++it) {

if ((\*it).barber == 0 || (\*it).barber == barber) {

bs.customers[(\*it).barber]--;

customer = \*it;

it.remove\_from\_list();

break;

}

}

}

region (customer) {

customer.startHaircut = true; // dodji u stolicu

await(!customer.startHaircut)

}

sleep(random()); // GIVE CUSTOMER A HAIRCUT

region (bs) {

customer.goHome = true; // isprati ga kuci

await(!customer.goHome);

}

}

// 2019 K

// 2.

monitor BankAccount {

int balance = 0;

condition c;

queue<int> amounts;

int balance() {

return balance;

}

void deposit(int amount) {

balance += amount;

while (balance - amounts.front() > 0) {

int d\_amount = amounts.pop();

balance -= amount;

c.signal();

}

}

void witdraw(int amount) {

if(c.queue() || balance - amount < 0) {

amounts.push\_back(amount);

int my\_turn = turn++;

c.wait(my\_turn);

}

else balance -= amount;

}

}

// 2019 Jul

// 1

class RollerCoaster {

int cart\_turn = 0;

int left\_spaces = 0;

int cart\_boarding = 0;

int cart\_to\_finish = 0;

cond waitForRide, waitBoarding, waitFull, waitUnboarding;

// int passenger\_turn = 0;

void ride() {

while (left\_spaces == 0)

waitForRide.wait();

left\_spaces -= 1;

if (left\_spaces == 0) waitFull.signal();

/\*

if (left\_spaces == 0)

waitForRide.wait(passenger\_turn++);

else {

left\_spaces -= 1;

if (left\_spaces == 0) waitFull.signal();

}

\*/

}

int boarding() {

int ride\_id = cart\_turn++;

if (ride\_id < cart\_boarding)

waitBoarding.wait(ride\_id);

left\_spaces = N;

waitForRide.signal\_all();

/\*

while (waitForRide.queue() && left\_spaces > 0){

left\_spaces--;

waitForRide.signal();

}

\*/

while (left\_spaces != 0)

waitFull.wait();

cart\_boarding += 1;

waitBoarding.signal();

return ride\_id;

}

void unboarding(int ride\_id) {

if (ride\_id > cart\_to\_finish)

waitUnboarding.wait(ride\_id);

cart\_to\_finish = ride\_id+1;

if (waitUnboarding.queue() && waitUnboarding.minrank() == cart\_to\_finish)

waitUnboarding.signal();

}

}

process car() {

boarding();

sleep(random()); //ride

unboarding();

}

process child() {

sleep(random()); // walk

ride();

}

// 2019. Jan

// 2.

monitor SwapMon {

int turn = 0;

int \*values[2];

int arrived = 0;

condition order, bar;

void swap(int \*val) {

int my\_turn = turn++;

if (arrived == 2)

order.wait(my\_turn);

arrived++;

values[my\_turn%2] = val;

if (arrived < 2)

bar.wait();

else {

swap(\*values[0], \*values[1]);

arrived = 0;

bar.signal();

order.signal();

order.signal();

}

}

}

// 2018. K2

// 1.

// Dovoljno je da bude jedan

monitor Queue {

bool busy = false;

condition waiting[3];

void new\_customer(int type) { // invalidi=0, majke=1, ostali=2

if (busy) {

waiting[type].wait(); // smatramo da je cv FIFO (inace preko 3 priority queue)

}

}

void finished() {

bool given = false;

for (int i = 0; i < 3; i++) {

if (waiting[i].queue()) {

waiting[i].signal();

given = true;

break;

}

}

if (!given) busy = false;

}

}

process client(int type, int account) {

int queue\_to\_wait = random() % QUEUE\_NUM;

queues[queue\_to\_wait].new\_customer(type);

int transaction = random() % 3;

switch (transaction) {

case 0: bank.deposit(account, random());

case 1: bank.withdraw(account, random());

case 2: bank.balance();

}

queues[queue\_to\_wait].finished();

}

// 2.

struct StudentDorm {

int portions = 0;

int cuttlery = 2;

int num\_eaten = 0;

int can\_eat[4] = {true, true, true, true}

};

shared StudentDorm sd;

void eat(int i) {

region(sd) {

await(sd.can\_eat[i]);

await(sd.cuttlery > 0);

sd.cuttlery--;

sd.waiting++;

await(sd.portions > 0);

sd.waiting--;

sd.portions--;

sd.num\_eaten++;

sd.can\_eat[i] = false;

}

sleep(random()); // eat

region (sd) {

sd.cuttlery++;

}

}

void cook() {

bool cookPortions = false;

region (sd) {

await(sd.num\_eaten == 4 || (sd.portions == 0 && sd.waiting > 0));

if (sd.num\_eaten == 4) {

sd.can\_eat[0] = sd.can\_eat[1] = sd.can\_eat[2] = sd.can\_eat[3] = true;

sd.num\_eaten = 0;

} else {

cookPortions = true;

}

}

if (cookPortions) {

sleep(random()); // prepare food

region (sd) {

sd.portions += K;

}

}

}

// Single Bathroom problem

const int K = 10;

struct Bathroom {

int man = 0, woman = 0, children = 0, janitor = 0;

int turn = 0, next = 0;

};

shared Bathroom b;

void enterMan() {

region (b) {

int my\_turn = turn++;

await(woman == 0 && janitor == 0 && next < my\_turn && (man + woman + children + janitor) < K);

next++;

man++;

}

}

void exitMan() {

region (b) {

await(children == 0 || man > 1);

man--;

}

}

// enterWoman, exitWoman

void enterChild() {

region (b) {

int my\_turn = turn++;

await(man + woman > 0 && next < my\_turn && (man + woman + children + janitor) < K);

next++;

child++;

}

}

void exitChild() {

region (b) {

child--;

}

}

void enterJanitor() {

region (b) {

int my\_turn = turn++;

await(man + woman + children + janitor > 0 && next < my\_turn && (man + woman + children + janitor) < K);

next++;

janitor++;

}

}

void exitJanitor() {

region (b) {

janitor--;

}

}

monitor Bathroom {

const int K = 20;

int turn = 0, next = 0;

int man=0, woman=0, children=0;

condition waitForTurn, waitExit;

void enterMan() {

int my\_turn = turn++;

if (my\_turn > next || (woman > 0 || man+children == K))

waitForTurn.wait(4\*my\_turn + 0);

next++;

man++;

regulate();

}

void exitMan() {

if (man == 1 && children > 0)

waitExit.wait();

man--;

regulate();

}

// ... enterWoman, exitWoman

void enterChild() {

int my\_turn = turn++;

if (my\_turn > next || (man+woman+children == K))

waitForTurn.wait(4\*my\_turn + 2);

next++;

children++;

regulate();

}

void exitChild() {

children--;

if (children == 0) {

waitExit.signal();

}

regulate();

}

void enterJanitor() {

int my\_turn = turn++;

if (my\_turn > next || (man+woman+children>0))

waitForTurn.wait(4\*my\_turn + 3);

}

void exitJanitor() {

next++;

regulate();

}

private:

void regulate() {

if (waitForTurn.queue()) {

int type = waitForTurn.minrank() % 4;

if (type == MAN && woman == 0 && (man+children)<K)

waitForTurn.signal();

// WOMAN

else if (type == CHILD && man+woman>0 && man+woman+children<K)

waitForTurn.signal();

else if (type == JANITOR && man + woman + children == 0)

waitForTurn.signal();

}

}

}

// 2017 K2

// 1. Taksi sluzba

// SC

class TaxiService {

int turn = 0;

condition waitForCar[3], waitForPassenger[3];

void want\_vehicle(int type) {

// ukoliko postoji tip koji trazi daj mu, ako ne, ceka (po tekstu ceka na

// STIZANJE svog ili veceg, iako nema logike)

if (waitForPassenger[type].queue()) {

waitForPassenger[type].signal();

} else {

int my\_turn = turn++;

waitForCar[type].wait(my\_turn);

}

}

void arrived(int type) {

// prednost imaju putnici koji traze bas taj tip,

// a ako njih nema onda popunjavamo one koji traze manje vozilo

// a ranije su dosli

if (waitForCar[type].queue()) {

waitForCar[type].signal();

} else {

int get\_from\_type = -1;

// pronadjem ko je prvi stigao od onih koji cekaju na manje vozilo od mog

for (int = 0; i < type; i++) {

if (waitForCar[i].queue() &&

(get\_from\_type == -1 || waitForCar[i].minrank() < waitForCar[get\_from\_type].minrank()))

get\_from\_type = i;

}

if (get\_from\_type == -1) {

waitForPassenger[type].wait();

} else {

waitForCar[get\_from\_type].signal();

}

}

}

}

// H2O monitori

// SW

monitor H2O {

int turn\_o = 0, next\_o = 0;

int turn\_h = 0, next\_h = 0;

condition h\_waiting, o\_waiting;

Molecule current;

Molecule oxygen(int id) {

int myTurn = turn\_o++;

if (myTurn > next\_o)

o\_waiting.wait(myTurn);

if (current.hydrogens != 2) {

ox\_waiting.wait();

}

current.addO(id);

Molecule my\_molecule = current;

molecule\_compiled.signal()

molecule\_compiled.signal();

next\_o++;

o\_waiting.signal();

return my\_molecule;

}

Molecule hydrogen(int id) {

int myTurn = turn\_h++;

if (myTurn > next\_h)

h\_waiting.wait(myTurn);

current.addH(id);

bool flag = false;

if(current.hydrogens == 2) {

flag = true;

ox\_waiting.signal();

} else {

next\_h++;

h\_waiting.signal();

}

if (!current.compiled())

molecule\_compiled.wait();

Molecule my\_molecule = current;

current.consumed++;

if (current.consumed == 2) {

current.reset();

}

if (flag) {

next\_h++;

h\_waiting.signal();

}

return my\_molecule;

}

};

// Barirera da svi znaju id ostalih

struct Info {

int id[N];

};

struct Barrier {

int cnt = 0;

bool open1 = true;

bool open2 = false;

Info info;

};

Info come(int id) {

Info ret;

region(b) {

await(open1);

info[cnt] = id;

cnt++;

if (cnt == N) {

open1 = false;

open2 = true;

}

await(open2);

cnt--;

ret = info;

if (cnt == 0) {

open2 = false;

open1 = true;

}

}

return ret;

}

// Dadilje (Childcare) regioni

struct Childcare {

int children = 0;

int workers = 0;

int turn = 0, next = 0;

};

shared Childcare cc;

void come\_to\_work() {

region (cc) {

workers++;

}

}

void go\_home() {

region (cc) {

int myTurn = turn++;

await(myTurn == next && children <= (workers-1)\*3);

workers--;

next++;

}

}

void bring\_children(int child\_num) {

region (cc) {

int myTurn = turn++;

await(myTurn == next && children + child\_num <= workers\*3);

children += child\_num;

next++;

}

}

Dadljie koje odlaze i roditelji koji dovode decu su medjusobno po FIFO redosledu kako bi se izbeglo izgladnjivanje.

// 2017 K2 2.

struct lift {

int goingOut[FLOORS];

int goingIn[FLOORS];

int floor;

enum phase;

};

process lift() {

int dir = 0;

while(true) {

region(lift) {

lift.floor += dir;

lift.phase = EXITING;

await(lift.goingOut[lift.floor] == 0);

lift.phase = ENTERING;

await(lift.goingIn[lift.floor] == 0);

dir = get\_direction();

lift.phase = RIDING;

}

ride(dir);

}

}

process rider() {

int currentFloor = ..., targetFloor = ...;

region(lift) {

lift.goingIn[currentFloor]++;

await(lift.floor == currentFloor);

await(lift.phase == ENTERING);

lift.goingIn[currentFloor]--;

}

ride\_in\_lift();

region (lift) {

lift.goingOut[targetFloor]++;

await(lift.floor == targetFloor);

lift.goingOut[targetFloor]--;

}

}

// 2020 Jun K2

3.

sem mutex = 1;

// svaka uslovna promenljiva ima brojac cnt\_cv i semafor cv

int cnt\_cv = 0; sem cv = 0;

// svaki monitor ima promenljivu owner (ko drzi monitor), cnt brojac ugnjezdenih poziva i sem enter koji obezbedjuje ekskluzivan pristup monitoru

ENTER:

wait(mutex);

if (owner != current\_thread) {

signal(mutex);

wait(enter);

wait(mutex);

owner = current\_thread;

signal(mutex);

cnt = 0;

} else signal(mutex);

cnt++;

EXIT:

cnt--;

if (cnt == 0)

signal(enter);

WAIT:

cv\_cnt++;

int myCnt = cnt;

signal(enter);

wait(cv);

wait(enter);

wait(mutex);

owner = current\_thread;

signal(mutex);

cnt = myCnt;

SIGNAL:

if (cv\_cnt > 0) {

cv\_cnt--;

signal(cv);

}

SIGNAL\_ALL:

while (cv\_cnt > 0) {

cv\_cnt--;

signal(cv);

}