Задачи за семафори

1. Всеки от процесите P и Q изпълнява поредица от три инструкции:

process P process Q

p\_1 q\_1

p\_2 q\_2

p\_3 q\_3

Осигурете чрез 2 семафора синхронизацията на P и Q така, че отделните инструкции да се изпълняват в реда: p\_1, q\_1, p\_2, q\_2, p\_3, q\_3

Използваме семафорите t1 и t2, инициализираме ги така:

semaphore t1, t2

t1.init(1)

t2.init(0)

Добавяме в кода на процесите P и Q синхронизиращи инструкции:

process P process Q

t1.wait() t2.wait()

p\_1 q\_1

t2.signal() t1.signal()

t1.wait() t2.wait()

p\_2 q\_2

t2.signal() t1.signal()

t1.wait() t2.wait()

p\_3 q\_3

t2.signal() t1.signal()

1. Множество паралелно работещи копия на всеки от процесите P и Q изпълняват поредица от 3 инструкции

process P process Q

p\_1 q\_1

p\_2 q\_2

p\_3 q\_3

Осигурете чрез семафори синхронизация на работещите копия, така че:

1.Три инструкции - p\_1, q\_2, p\_3 се редуват циклично.

2.Първа се изпълнява инструкция p\_1 на някое от работещите копия на процес P. След завършването й се изпълнява инструкция q\_2 на някое копие на Q, а след нея - p\_3 на копие на P. С това цикълът завършва и отново може да се изпълни инструкция p\_1 на някое от работещите копия на процес P.

За исканите в условието синхронизации използваме три семафора - t1, t2 и t3, инициализираме ги така:

semaphore t1, t2, t3

t1.init(1)

t2.init(0)

t3.init(0)

Добавяме в кода на процесите P и Q синхронизиращи инструкции:

process P process Q

t1.wait() q\_1

p\_1 t2.wait()

t2.signal() q\_2

p\_2 t3.signal()

t3.wait() q\_3

p\_3

t1.signal()

1. Множество паралелно работещи копия на всеки от процесите P и Q изпълняват поредица от 2 инструкции:

process P process Q

p\_1 q\_1

p\_2 q\_2

Осигурете чрез семафори синхронизация на работещите копия, така че:

1.Инструкцията q\_2 на всяко от работещите копия на Q да се изпълни след като инструкция p\_1 е завършила изпълнението си в поне 3 работещите копия на P. Упътване: Освен семафори, ползвайте и брояч.

За исканите в условието синхронизации използваме брояч cnt и 2 семафора - m1 и m2, инициализираме ги така:

semaphore m1, m2

m1.init(1)

m2.init(0)

int cnt=0

Добавяме в кода на процеса P синхронизиращи инструкции:

process P

p\_1

m1.wait()

cnt=cnt+1

if cnt=3 m2.signal()

m1.signal()

p\_2

Добавяме в кода на процеса Q синхронизиращи инструкции:

process Q

q\_1

m2.wait()

m2.signal()

q\_2

Семафорът m1 ползваме като мутекс, който защитава брояча. Стойността на cnt е равна на броя копия на процеса P, които са изпълнили своята първа инструкция. Семафорът m2 блокира изпълнението на инструкция q\_2. Когато третото копие на процеса P изпълни p\_1, към семафора m2 се подава сигнал, който го деблокира и позволява на всички копия на Q да изпълнят втората си

инструкция.

1. Множество паралелно работещи копия на всеки от процеса P изпълняват поредица от 2 инструкции:

process P

p\_1

p\_2

Осигурете чрез семафори синхронизация на работещите копия, така че:

1.Инструкцията p\_2 на всяко от работещите копия да се изпълни след като инструкция p\_1 е завършила изпълнението си в поне 3 работещи копия. Упътване: Освен семафори, ползвайте и брояч.

За исканите в условието синхронизации използваме брояч cnt и 2 семафора - m1 и m2, инициализираме ги така:

semaphore m1, m2

m1.init(1)

m2.init(0)

int cnt=0

Добавяме в кода на процеса P синхронизиращи инструкции:

process P

p\_1

m1.wait()

cnt=cnt+1

if cnt=3 m2.signal()

m1.signal()

m2.wait()

m2.signal()

p\_2

Семафорът m1 ползваме като мутекс, който защитава брояча. Стойността на cnt е равна на броя копия на процеса P, които са изпълнили своята първа инструкция. Семафорът m2 блокира изпълнението на инструкция p\_2. Когато третото копие на процеса P изпълни p\_1, към семафора m2 се подава сигнал, който го деблокира и позволява на всички копия да изпълнят втората си инструкция.

1. Всеки от процесите P, Q и R изпълнява поредица от три инструкции:

process P process Q process R

p\_1 q\_1 r\_1

p\_2 q\_2 r\_2

p\_3 q\_3 r\_3

Осигурете чрез семафори синхронизация на P, Q и R така, че да се изпълнят следните изисквания:

(а) Инструкция p\_1 да се изпълни преди q\_2 и r\_2.

(б) Ако q\_2 се изпълни преди r\_2, то и q\_3 да се изпълни преди r\_2.

(в) Ако r\_2 се изпълни преди q\_2, то и r\_3 да се изпълни преди q\_2.

За синхронизация използваме семафор t, инициализираме го с блокиращо начално състояние :

semaphore t

t.init(0)

Добавяме в кода на процесите P, Q и R синхронизиращи инструкции:

process P process Q process R

p\_1 q\_1 r\_1

t.signal() t.wait() t.wait()

p\_2 q\_2 r\_2

p\_3 q\_3 r\_3

t.signal() t.signal()

Всяка от инструкциите q\_2 и r\_2 може да се изпълни след като броячът на семафора t стане положителен. Това се случва за пръв път след изпълнението на ред t.signal() в процеса P, който следва инструкция p\_1. Така гарантираме изпълнението на изискване (а). След като броячът на семафора стане 1, един от процесите Q и R ще достигне пръв до ред t.wait() и ще го нулира отново. Ако процесът Q пръв достигне инструкцията t.wait(), той ще изпълни редове q\_2 и q\_3, а процесът R ще чака ново увеличение на брояча на семафора, което се случва след изпълнението на последния ред t.signal() в процеса Q. Така гарантираме изпълнението на изискване (б). Ако процесът R пръв достигне инструкцията t.wait(), той ще изпълни редове r\_2 и r\_3, а процесът Q ще чака ново увеличение на брояча на семафора, което се случва след изпълнението на последния ред t.signal() в процеса R. Така гарантираме изпълнението на изискване (в).

1. Всеки от процесите P и Q изпълнява поредица от две инструкции:

process P process Q

p\_1 q\_1

p\_2 q\_2

Осигурете чрез семафори синхронизацията на P и Q така, че инструкция p\_1 да се изпълни преди q\_2, а q\_1 да се изпълни преди p\_2.

За двете искани в условието синхронизации използваме два семафора – t1 и t2, инициализираме ги с блокиращо начално състояние:

semaphore t1, t2

t1.init(0)

t2.init(0)

Добавяме в кода на процесите P и Q синхронизиращи инструкции:

process P process Q

p\_1 q\_1

t1.signal() t2.signal()

t2.wait() t1.wait()

p\_2 q\_2

Инструкцията q\_2 ще се изпълни след като процесът Q премине бариерата t1.wait(). Това се случва след изпълнението от P на ред t1.signal(), койо следва инструкция p\_1. Аналогично, инструкцията p\_2 ще се изпълни след изпълнението на ред t2.signal(), който следва инструкцията q\_1. Решението на задачата осигурява среща във времето (rendezvous) на двата процеса. Важен е редът на извикване на инструкциите, управляващи семафорите. Ако го обърнем, получаваме класически пример за deadlock.

1. Паралелно работещи копия на всеки от процесите P и Q изпълняват поредица от 2 инструкции

process P process Q

p\_1 q\_1

p\_2 q\_2

Осигурете чрез семафори синхронизацията на работещите копия така че:

а) Във произволен момент от времето да работи най-много едно от копията.

б) Работещите копия да се редуват във времето – след изпълнение на копие на P, да следва изпълнение на копие на Q, и обратно.

в) Първоначално да е разрешено да се изпълни копие на P.

Използваме два семафора – s\_p и s\_q, инициализираме ги така:

semaphore s\_p, s\_q

s\_p.init(0)

s\_q.init(0)

Добавяме в кода на процесите P и Q синхронизиращи инструкции:

process P process Q

s\_p.wait() s\_q.wait()

p\_1 q\_1

p\_2 q\_2

s\_q.signal() s\_p.signal()

1. Няколко копия на процеса P изпълняват поредица от три инструкции:

process P

p\_1

p\_2

p\_3

Осигурете чрез семафор синхронизацията на копията така, че най-много един процес да изпълнява инструкция p\_2 във всеки един момент.

1. Всеки от процесите P, Q и R изпълнява поредица от три инструкции:

process P process Q process R

p\_1 q\_1 r\_1

p\_2 q\_2 r\_2

p\_3 q\_3 r\_3

Осигурете чрез семафори синхронизация на P, Q и R така, че да са изпълнени едновременно условията:

1. инструкция p\_1 да се изпълни преди q\_2 и r\_2.
2. инструкция p\_3 да се изпълни след q\_2, r\_2.

Добавяме в кода на процесите P, Q и R синхронизиращи инструкции:

process P process Q process R

p\_1 q\_1 r\_1

s.signal() s.wait() s.wait()

p\_2 s.signal() s.signal()

t.wait() q\_2 r\_2

u.wait() t.signal() u.signal()

p\_3 q\_3 r\_3

Всяка от инструкциите q\_2 и r\_2 може да се изпълни след като съответния процес премине бариерата s.wait(). Това се случва за пръв път след изпълнението на ред s.signal() в процеса P, който следва инструкция p\_1. Така изпълнението на p\_1 преди q\_2 и r\_2 е гарантирано. Да допуснем, че процесът Q преминава през инструкцията си s.wait() преди процеса R. Веднага след това той изпълнява s.signal(), което ще позволи и на R да премине през своята инструкция s.wait(). Така ще се осигури изпълнението и на двете инструкции q\_2 и r\_2. Аналогична е ситуацията, когато R преминава през s.wait() преди процеса Q. Работата със семафорите t и u осигурява изпълнението на условие (2).

1. Всеки от процесите P, Q и R изпълнява поредица от три инструкции:

process P process Q process R

p\_1 q\_1 r\_1

p\_2 q\_2 r\_2

Осигурете чрез три семафора синхронизацията на P, Q и R така, че отделните инструкции да се изпълнят в следния времеви ред: p\_1, q\_1, r\_1, p\_2, q\_2, r\_2

Използваме семафорите t1, t2 и t3, инициализираме ги така:

semaphore t1, t2, t3

t1.init(1)

t2.init(0)

t3.init(0)

Добавяме в кода на процесите синхронизиращи инструкции:

process P process Q process R

t1.wait() t2.wait() t3.wait()

p\_1 q\_1 r\_1

t2.signal() t3.signal() t1.signal()

t1.wait() t2.wait() t3.wait()

p\_2 q\_2 r\_2

t2.signal() t3.signal() t1.signal()

1. Всеки от процесите P и Q изпълнява поредица от три инструкции:

process P process Q

p\_1 q\_1

p\_2 q\_2

p\_3 q\_3

Осигурете чрез два семафора синхронизацията на P и Q, така че да са изпълнени едновременно следните времеви зависимости:

1. Инструкция p\_1 да се изпълни преди q\_2
2. Инструкция q\_2 да се изпълни преди p\_3
3. Инструкция q\_1 да се изпълни преди p\_2
4. Инструкция p\_2 да се изпълни преди q\_3

Условия (1) и (3) определят времева среща (randevous) на процесите след първата им инструкция. Аналогично (2) и (4) определят randevouz на процесите след втората им инструкция. За двете срещи използваме два семафора – t1 и t2, инициализираме ги с блокиращо начално състояние:

semaphore t1, t2

t1.init(0)

t2.init(0)

Добавяме в кода на процесите P и Q синхронизиращи инструкции:

process P process Q

p\_1 q\_1

t1.signal() t2.signal()

t2.wait() t1.wait()

p\_2 q\_2

t1.signal() t2.signal()

t2.wait() t1.wait()

p\_3 q\_3

Инструкцията q\_2 ще се изпълни след като броячът на семафора t1 стане положителен. Това се случва след изпълнението на ред t1.signal(), който следва инструкция p\_1. Аналогично, инструкцията p\_2 ще се изпълни след като броячът на семафора t2 стане положителен. Това се случва след изпълнението на ред t2.signal(), който следва инструкция q\_1. По подобен начин ще се развият събитията и след вторите инструкции. Лесно се вижда, че след първото randevouz стойностите на броячите в семафорите ще са 0 и процесите коректно ще реализират втората среща със същите семафори.

1. Множество паралелно работещи копия на всеки от процесите P и Q изпълняват поредица от две инструкции:

process P process Q

p\_1 q\_1

p\_2 q\_2

Осигурете чрез семафори синхронизация на P и Q, така че поне една инструкция p\_1 да се изпълни преди всички q\_2, и поне една инструкция q\_1 да се изпълни преди всички p\_2.

За двете искани в условието синхронизации използваме два семафора – t1 и t2, инициализираме ги с блокиращо начално състояние:

semaphore t1, t2

t1.init(0)

t2.init(0)

Добавяме в кода на процесите P и Q синхронизиращи инструкции:

process P process Q

p\_1 q\_1

t1.signal() t2.signal()

t2.wait() t1.wait()

t2.signal() t1.signal()

p\_2 q\_2

Произволна инструкция q\_2 ще се изпълни, след като изпълняващото я копие на процеса Q премине бариерата t1.wait(). Бариерата ше се отпуши след изпълнението от поне едно копие на P на ред t1.signal(), който следва инструкция p\_1. Копията на Q изпълняват поредица t1.wait(), t1.signal(). Така семафорът t1 ще събуди всички приспани други копия на Q и ще осигури завършването им. Аналогично, произволна инструкция p\_2 ще се изпълни след първото изпълнение на ред t2.signal() в някое копие на Q, който следва инструкция q\_1.

1. Всеки от процесите P, Q и R изпълнява поредица от три инструкции:

process P process Q process R

p\_1 q\_1 r\_1

p\_2 q\_2 r\_2

p\_3 q\_3 r\_3

Осигурете чрез семафори синхронизация на P, Q и R така, че да изпълняват следните изисквания:

(а) инструкция p\_1 да се изпълни преди q\_2 и r\_2.

(б) инструкция r\_2 да се изпълни след p\_3.

1. Всеки от процесите P, Q и R изпълнява поредица от три инструкции:

process P process Q process R

p\_1 q\_1 r\_1

p\_2 q\_2 r\_2

p\_3 q\_3 r\_3

Осигурете чрез семафори синхронизация на P, Q и R така, че да се изпълнят едновременно следните изисквания:

(а) Някоя от инструкциите p\_2 и q\_2 да се изпълни преди r\_2.

(б) Ако инструкция p\_2 се изпълни преди r\_2, то q\_2 да се изпълни след r\_2.

(в) Ако инструкция q\_2 се изпълни преди r\_2, то p\_2 да се изпълни след r\_2.

За синхронизация използваме семафори f и u, инициализираме ги така:

semaphore f, u

f.init(1)

u.init(0)

Добавяме в кода на процесите P, Q и R синхронизиращи инструкции:

process P process Q process R

p\_1 q\_1 r\_1

f.wait() f.wait() u.wait()

p\_2 q\_2 r\_2

u.signal() u.signal() f.signal()

p\_3 q\_3 r\_3

Инструкция r\_2 може да се изпълни след като семафорът u, който в началото е блокиран, получи сигнал. Това става единствено след изпълнението на някоя от инструкциите p\_2 и q\_2. Така осигуряваме изпълнението на условие (а). Броячът на семафора f в началото е 1, само един от процесите P и Q ще премине реда си f.wait() и ще го нулира, другият процес ще чака сигнал. Това става само след изпълнението на ред f.signal() от процеса R, след изпълнение на инструкция r\_2. Така осигуряваме изпълнението на условия (б) и (в). Ако процесът P пръв достигне инструкцията f.wait(), ще се изпълни предпоставката на условие (б), редът на изпълнение на интересните инструкции ще е p\_2, r\_2, q\_2. Ако процесът Q пръв достигне инструкцията f.wait(), ще се изпълни предпоставката на условие (в), редът на изпълнение на интересните инструкции ще е q\_2, r\_2, p\_2.

Теоретични задачи

* **Опишете накратко основните процедури и структури от данни, необходими за реализация на семафор.**

Структурите данни, необходими за реализация на семафор са:

1.Брояч cnt, в който се пази броя на процесите, които могат да бъдат допуснати до ресурса, охраняван от семафора.

2.Контейнер Q, в който се пази информация кои процеси чакат достъп до ресурса.

**Процедурите, необходими за реализация на семафор са:**

1.Конструктор Init(c0:integer), който задава начална стойност на брояча cnt. Контейнерът Q се инициализира да е празен.

2.Метод Wait(), който се ползва при опит за достъп до ресурса(заемане на ресурса). Броячът се намалява с 1 и ако стане отрицателен, процесът викащ Wait() се блокира, а номерът му се вкарва в контейнера Q.

3.Метод Signal(), който се ползва при завършване на достъпа до ресурса(освобождаване на ресурса). Броячът се увеличава с 1 и ако Q не е празен, един от процесите в него се вади и активира

**Каква е разликата между слаб и силен семафор?**

Семафор е силен, когато контейнера Q е реализиран като обикновена опашка - винаги активираме процеса, блокиран най-рано.

Семафор е слаб, когато контейнера Q не е реализиран като обикновена опашка - при изпълнение на Signal() активираме процес, който може да не е първи в списъка на чакащите.

**Опишете максимално несправедлива ситуация, която може да се получи в избирателна секция, ако на входа на секцията пазач - член на изборната комисия пуска гласоподавателите вътре така:**

**1. Във всеки момент в секцията може да има най-много двама гласоподаватели.**

**2. Пазачът работи като слаб семафор.**

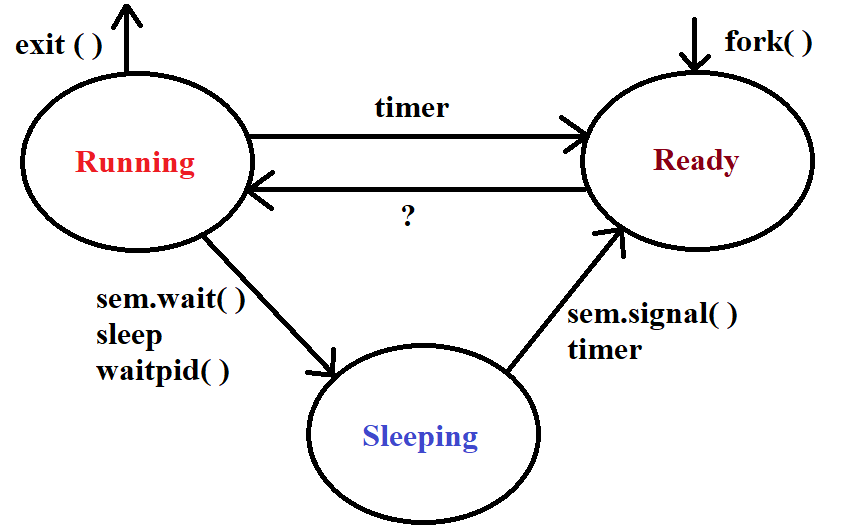
Ако пазачът на входа на избирателната секция действа като слаб семафор, може да се получи ситуацията:

Първите двама гласоподаватели влизат в секцията, пристига трети гласоподавател(неприятел на пазача) и чака. След него пристигат приятели на пазача и той ги пуска с предимство. Така третият гласоподавател чака цял ден и гласува последен. Такава ситуация при достъп до ресурс се нарича starvation (гладуване).

* **Какви са възможните състояния на процес. Нарисувайте диаграма на състоянията и преходите между тях. Опишете накратко ситуациите, предизвикващи преходи между състоянията.**

Възможните състояния на процес са:

* Running – използва активно CPU
* Ready – очаква CPU
* Sleeping – приспан/блокиран процес, очаква вход-изход
* Stopped – процес в периода между kill и освобождаването на ресурс



* **Опишете разликата между синхронни и асинхронни входно-изходни операции. Дайте примери за програми, при които се налага използването на асинхронен вход-изход.**

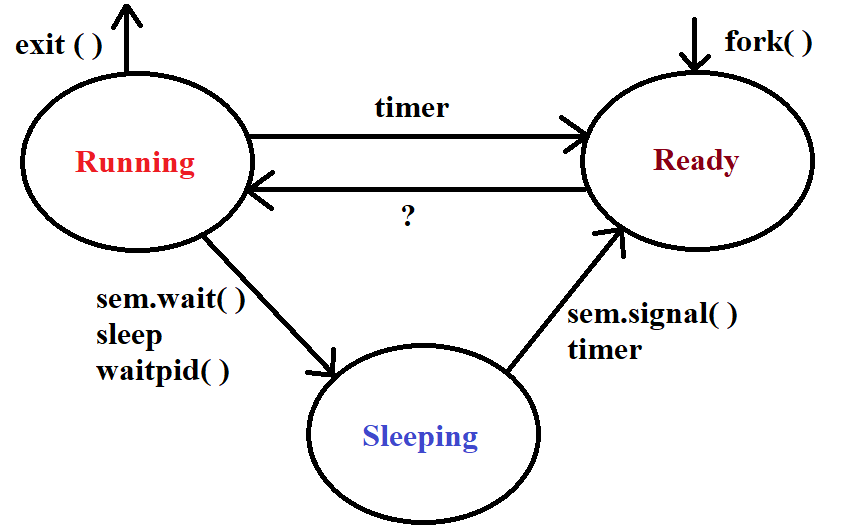
При синхронна входно-изходна операция системното извикване може да доведе до приспиване (блокиране) на потребителския процес, поръчал операцията. Същевременно, при нормално завършване, потребителският процес разчита на коректно комлектоване на операцията – четене/запис на всички предоставени/поръчани данни във/от входно-изходния канал, или цялостно изпълнение на друг вид операция (примерно, изграждане на TCP връзка). При асинхронна входно-изходна операция системното извикване не приспива (не блокира) потребителския процес, поръчал операцията. Същевременно, при невъзможност да се комплектова операцията, ядрото връща управлението на процеса със специфичен код на грешка и друга информация, която служи за определяне на степента на завършеност на операцията. Потребителският процес трябва да анализира ситуацията и при нужда да направи ново системно повикване по-късно, с цел да довърши операцията. Използването на асинхронни операции позволява на един процес да извършва паралелна комуникация по няколко канала с различни устройства или процеси, без да бъде блокиран в случай на липса на входни данни, препълване на буфер за изходни данни или друга ситуация, водеща до блокиране. Типични примери: (1) Когато ползваме WEB-browser, той трябва да реагира на входни данни от клавиатура и мишка, както и на данните, постъпващи от интернет, т.е. на поне 3 входни канала. Браусерът проверява чрез асинхронни опити за четене по кой от каналите постъпва информация и реагира адекватно. (2) Сървер в интернет може да обслужва много на брой клиентски програми, като поддържа отворени TCP връзки към всяка от тях. За да обслужва паралелно клиентите, сърверът трябва да ползва асинхронни операции, за да следи по кои връзки протича информация и кои са пасивни. Когато програмата ползва асинхронни операции и никой от входно-изходните канали не е готов за обмен на данни, тя има нужда от специален механизъм за предоствяне на изчислителния ресурс на останалите процеси. Обикновено в такива случаи програмата се приспива сама за кратък период от време (в UNIX това става с извикване на sleep(), usleep() или nanosleep()).

* **Опишете какви изисквания удовлетворява съвременна файлова система, реализирана върху блочно устройство (block device). Опишете накратко реализацията и целта на следните инструменти:**

**(а) отлагане на записа, алгоритъм на асансьора**

**(б) поддържане на журнал на файловата система**

* **Опишете накратко кои системни извиквания изграждат стандартните комуникационни канали в UNIX – неименувана тръба (pipe), връзка процес-файл, двустранна връзка процес-процес (connection).**
* **Опишете реализацията на комуникационна тръба (pipe) чрез семафори. Предполагаме, че тръбата може да съхранява до n байта, подредени в обикновена опашка. Тръбата се ползва от няколко паралелно работещи изпращачи/получатели на байтове. Процесите изпращачи слагат байтове в края на опашката, получателите четат байтове от началото на опашката. Упътване: В теорията на конкурентното програмиране задачата е известна като "producer-consumer problem".**
* **Да приемем, че в съвременната операционна система процесът има 4 състояния: R – работещ (running, използва CPU) A – активен (ready, очаква CPU) S – блокиран (sleeping, очаква вход/изход) T – изчакващ време (sleeping, очаква времеви момент). Нарисувайте диаграма на състоянията и преходите между тях. Диаграмата е ориентиран граф с върхове отделните състояния и ребра – възможните преходи. Опишете накратко събитията, предизвикващи преход по всяко ребро на графа.**



* **Опишете накратко основните комуникационни канали в ОС Linux. Кои канали използват пространството на имената и кои не го правят?**

Основните комуникационни канали в ОС Linux са тръба (pipe), именувана тръба (fifo), връзка процес-файл и конекция (изградена с механизма socket). Всички, освен обикновената тръба, използват пространството на имената. Именуваната тръба се ползва рядко, ако не я споменете, не губите точки. Операциите за ползване на канала са общи – read(…), write(…), close(…). Специфични са извикванията за изграждане на различните видове канали.

* **Опишете как се изгражда комуникационен канал (connection) между процессървер и процес-клиент със следните системни извиквания в стандарта POSIX: socket(), bind(), connect(), listen(), accept()**
* **Опишете реализацията на комуникационна тръба (pipe) чрез семафори. Предполагаме, че тръбата може да съхранява до n байта, подредени в обикновена опашка. Тръбата се ползва от няколко паралелно работещи изпращачи/получатели на байтове. Процесите изпращачи слагат байтове в края на опашката, получателите четат байтове от началото на опашката.**