Асиметрични протокол извртљивог бита

Април 2004

1 Увод

Протокол извртљивог бита (енг. *Alternating Bit Protocol*) је добро познат. Али, то је симетрични протокол, за два равноправна учесника комуникације.

Протокол који овде описујемо је, практично, асиметрична варијанта протокола извртљивог бита. Користи се када један од учесника прозива (енг. *polling*) другог. Тај корисник се обично назива **газда** (енг. *master*). Дакле, биће потребно да се посматра и са стране газде на магистрали и слуге на магистрали.

Један газда може да комуницира преко једне магистрале са више слуга. За сваког слугу се овај протокол води посебно.

У опису који следи, провера исправности поруке ради се путем ЦРЦ кода али то је само уобичајени начин провере који је коришћен у пракси. Било који други начин (рецимо, ИП контролни збир) је могућ, протокол се тиме не бави.

2 Из угла газде

Када први пут треба да прозове слугу, газда поставља ФИБ бит на 1.

Када год треба да прозове слугу, газда проверава да ли има поруке за дотичног. Ако има, шаље их. Ако порука нема, шаље поруку попуне (празну поруку) и наравно, шаље ФИБ бит какав је наумио.

Након слања, газда чека одговор слуге.

Ако одговор не стигне, или стигне са грешком (ЦРЦ грешком, пре свега), ФИБ бит остаје ”где је и био” и поруке које су биле послате неће бити ”ослобођене”. Дакле, биће поново послате при следећој прозивци. Наравно, неисправна порука се одбацује.

Ако одговор стигне, проверава се да ли је добијен ФИБ бит исти или различит од послатог. Ако је исти, као да одговор није ни стигао за сврхе овог протокола (за дијагностичке сврхе, слуга је ”жив”).

Ако је примљени бит другачији (изврнут) од послатог, онда се сматра да је порука исправно примљена на другој страни и послате поруке се ослобађају (неће бити поново слате). Газда изврће ФИБ бит за тог слугу за следећи пут.

Тиме се завршава прозивка једног слуге и прелази на следећег. Када се све слуге потроше, враћа се на првог и тако укруг.

За сврхе дијагностике, ако слуга не одговори на прозивку неколико пута заредом, пријављује се његов отказ.

3 Из угла слуге

У иницијализацији, слуга поставља ”свој” ФИБ бит на 1.

Ако успешно прими поруку (са исправним ЦРЦ-ом), слуга проверава да ли је примљени ФИБ бит исти или различит од ”његовог” ФИБ бита.

Ако је примљени ФИБ бит исти као очекивани, значи да стиже нова порука и да је наша порука прихваћена. Слуга ослобађа поруке које је послао прошли пут и изврће ФИБ бит. Затим шаље нашу поруку и уписује тај (изврнути) ФИБ бит. Ако нема поруке, шаље поруку попуне (празну поруку).

Ако је примљени ФИБ бит различит од очекиваног, значи да газда није исправно примио поруку прошли пут. Примљена порука се занемарује, а слуга поново шаље исту поруку, са истим ФИБ битом.

Ако је порука примљена са ЦРЦ грешком, поступа се слично - шаље се поново иста порука као малопре, са истим ФИБ битом (неисправна порука се, наравно, занемарује).

4 Дискусија

Ако се ФИБ бит постави на 0 у иницијализацији слуге, алгоритам пати од губљења прве поруке. Али, ваља обратити пажњу - то може и не мора да буде тачно, односно, не помаже много да слуга свој ФИБ бит у иницијализацији постави на 1. Наиме, та иницијализација је асинхрона у односу на газдину иницијализацију.

Због тога, газда треба, први пут када прозива слугу, да пошаље поруку попуне - јер ако се изгуби та, празна, порука, неће бити губљења ”стварне” поруке.

Узгред, ни то не решава ствар у потпуности. Пошто се у иницијализацији ради све и свашта, непознато дуго, а за све то време газда може да прозива, опет ће се десити да се нека порука ту и тамо изгуби. То би, теоријски, могло да се среди, али, претпоставка је да се слуге релативно ретко иницијализују.

5 СПИН модел протокола

У наставку је Промела код СПИН модела овог протокола. Треба га снимити у засебну датотеку, рецимо, `FIB_BIT_KOM` (датотеке са Промела изворним кодом традиционално немају наставак). Када се тако добијена датотека ”пусти” кроз проверавач модела СПИН, може да се види да је протокол ”ваљан”.

Код је писан у жаргону система СРЦЕ, па је КОП газда, док је РП слуга у смислу протокола који описујемо.

Модел садржи три процеса:

1. КОП - газда
2. РП - слуга (само један, јер је протокол са свима исти)
3. магистрала, која служи да симулира грешке

СПИН проверавачем може да се утврди да протокол правилно преноси поруке (редом) и да се ”не заглављује”. Да би се избегло пријављивање бесконачног губљења порука (ако

је магистрала толико лоша да губи све поруке и ниједну не преноси) као ”заглављивање”, одговарајући случајеви су обележени са `progress...` лабелама.

```
/** Maksimalan broj poruke - treba da bude bar dva. */
#define MAX    10

/** Pomocna definicija za "neispravan CRC" - radi citljivosti */
#define BAD_CRC  0

/** Pomocna definicija za "ispravan CRC" - radi citljivosti */
#define GOOD_CRC 1

/** KOP, odnosno "gazda" na magistrali, proziva RP-ove, odnosno
    "sluge" na magistrali. Za svrhe ovog modela, samo jedan RP
    (za FIB bit protokol RP-ovi su nezavisni).
    @param in Kanal po kome KOP prima poruke
    @param out Kanal po kome KOP predaje poruke
    */
proctype kop(chan in, out)
{
    byte by; /* Poruka za slanje - za probne svrhe, broj */
    bit fib; /* "Nas" FIB */
    bit fib_prim; /* FIB u primljenoj poruci - odgovoru */
    byte by_prim; /* Primljena poruka - odgovor */

    by = MAX - 1; /* U prvom prolasku kroz petlju postane 0 */
    fib = 1;

    do
    :: by = (by + 1) % MAX;
prozivka:
        out!fib, by, GOOD_CRC;

        if
        :: timeout -> goto prozivka /* nema odgovora */
        :: in?fib_prim, by_prim, BAD_CRC -> goto prozivka
        :: in?fib_prim, by_prim, GOOD_CRC ->
            if
            :: (fib == fib_prim) -> goto prozivka
            :: (fib != fib_prim) ->
                assert(by_prim == (by + 1) % MAX);
                goto progress
            fi
        fi;
progress:
        fib = !fib
    od
}

/** Proces RP-a na magistrali. RP odgovara na prozivku. Ovo nije
```

```
    potpuno veran model, jer RP poruke ulancava u red za obradu,
    pa odgovor na prozivku ne bude odgovor na datu poruku, ali,
    to je nevazno - "poruke" koje u ovom modelu razmenjujemo su
    samo pomocni  brojaci da se uverimo da nece doci do gubitaka
    ili dupliranja poruke.
    @param in Kanal po kome RP prima poruke
    @param out Kanal po kome RP predaje poruke
    */
proctype rp(chan in, out)
{
    bit fib; /* "Nas FIB" */
    byte by; /* Poruka za prijem - za probu, broj */
    bit fib_prim; /* FIB primljene poruke */
    byte by_prim; /* Primljena poruka */

    fib = 1;
    by = 0;

    do
    :: in?fib_prim,by_prim,BAD_CRC -> out!fib,by,GOOD_CRC
    :: in?fib_prim,by_prim,GOOD_CRC ->
        if
        :: (fib_prim != fib) -> out!fib,by,GOOD_CRC
        :: (fib_prim == fib) ->
            assert(by_prim == by);
            by = (by + 1) % MAX;
progress:   fib = !fib;
            out!fib,by,GOOD_CRC;
        fi
    od
}


/** Magistrala. Slucajno unosi greske - gubitak poruke i gresku
    koja dovodi do loseg CRC-a. Naravno, ovako ne mogu da se
    simuliraju greske koje CRC  ne uhvati, ali ovaj model i ne
    odredjuje koji CRC se koristi. Sto bolji  CRC, to ce vise
    gresaka biti uhvaceno i bice manje gresaka u prenosu.

    @param kop_in Kanal po kome KOP prima poruke (a RP salje)
    @param kop_out Kanal po kome KOP salje poruke (a RP prima)
    @param rp_in Kanal po kome RP prima poruke (a KOP salje)
    @param rp_out Kanal po kome RP salje poruke (a KOP prima)
    */
proctype magistrala(chan kop_in, kop_out, rp_in, rp_out)
{
    bit fib;
    byte by;
    bit crc;
```

```
    do
    ::kop_out?fib,by,crc ->
        if
        ::rp_in!fib,by,GOOD_CRC
        ::rp_in!fib,by,BAD_CRC; progress_sgreskom1:skip;
        ::skip; progress_gubitak1:skip;
        fi
    ::rp_out?fib,by,crc ->
        if
        ::kop_in!fib,by,GOOD_CRC
        ::kop_in!fib,by,BAD_CRC; progress_sgreskom2:skip;
        ::skip; progress_gubitak2:skip;
        fi
    od
}


/** Inicijalizacija modela */
init {
    chan kop_predaja = [1] of {bit, byte, bit};
    chan kop_prijem = [1] of {bit, byte, bit};
    chan rp_predaja = [1] of {bit, byte, bit};
    chan rp_prijem = [1] of {bit, byte, bit};

    atomic {
        run kop(kop_prijem, kop_predaja);
        run magistrala(kop_prijem, kop_predaja, rp_prijem,
            rp_predaja);
        run rp(rp_prijem, rp_predaja);
    }
}
```