Magasszintű Adatkapcsolati Protokoll (<u>H</u>igh Level <u>D</u>ata <u>L</u>ink <u>C</u>ontrol Protocol - HDLC)

összeállította: Tőke Pál

April 2, 1999

1 Bezezetés

2 HDLC állomás komponenseinek áttekintése

2.1 <u>Kapcsolatfelépítő</u> *(Link-set-up)* komponens

Verziók:

- elsődleges állomás,
- másodlagos állomás.

Lehetséges transzformációk:

- Elsődleges állomás: $\underline{SXRM}$, $\underline{DISC}$, UA, $CMDR$, $ERROR$.
- Másodlagos állomás: $SXRM$, $DISC$ $\underline{UA}$, $\underline{CMDR}$.

2.2 <u>Küldő</u> *(Source)* komponens

Lehetséges transzformációk: $\underline{I}$, RNR, RR, REJ.

Változói: $buffer$, VS, $unack$.

2.3 <u>Fogadó</u> *(Sink)* komponens

Verziók:

- fogadás visszautasítás (REJ) nélkül,
  fogadás visszautasítás (REJ) lehetőségével.

Lehetséges transzformációk: I, $\underline{RNR}$, $\underline{RR}$, $\underline{REJ}$.

Változói: $buffer$, VR.

2.4 <u>P/F-kontroll</u> *(PF-control)* komponens

Verziók:

- elsődleges normál válaszüzemmódban (NRM),
- elsődleges aszinkron válaszüzemmódban (ARM),
- másodlagos normál válaszüzemmódban (NRM),
- másodlagos aszinkron válaszüzemmódban (ARM).

Lehetséges transzformációk:

- Elsődleges állomás: $\underline{P_0}$, $\underline{P_1}$, F_0, F_1, $\underline{P_{1-ism}}$.
  Másodlagos állomás: P_0, P_1,$\underline{F_0}$, $\underline{F_1}$.

Változó: bit.

2.5 <u>Óra</u> *(Clock)* komponens

Lehetséges transzformáció: $TICK$.

Változó: $timer$.

2.6 <u>Átvitel</u> *(Transmission)* komponens

Változók: $received$, $status$

Lehetséges távakció: $transmit$.

Távolról kezdeményezett transzformáció: $transmit$.

2.7 <u>Ellenőrzőpont</u> *(Checkpoint)* komponens

Változók: VS, $inhibit$ (az állomás típusától függően).

3 Adattípusok és konstansok

Konstansok	Jelentés
modulus = 8;	Az *I* keretek számlálójának modulusa

Típusok	Jelentés
info-type = ... ;	Az információ mező számára a keretekben
address-type = ... ;	A címek számára a keretekben
sequence-count = 0 .. modulus - 1;	
frame-kind = (none, I, RR, RNR, REJ, SARM, SNRM, DISC, UA, CMDR);	
control-type = record *kind: frame-kind;* *pfbit: (0 .. 1);* *NS: sequence-count;* *NR: sequence-count;* *end;*	HDLC keretek kontroll mezőinek elemei
frame-type = record *address: address-type;* *control-field: control-type;* *info: info-type;* *end;*	HDLC keretek típusa
status-type = set of [*invalid-control-field,* *invalid-info,* *invalid-size,* *invalid-NR,* *time-out* *];*	A *másodlagos* állomás által jelezhető hibák

4 Átvitel komponens

Változók	Jelentés
received : frame-type	A partner állomásnál kezdeményezett *transmit* éredményeként fogadott keretet tartalmazza
status : status-type	A fogadott keret állapota, vagy *timeout*

Kezdeti állapot

status = [];
received.kind = none;

Eljárások

examine-NR;

{ Ez az eljárás felszabadítja a *buffer*t és beállítja *Küldő* komponens változóit a *received.NR*-nek megfelelően }

```
begin
    if received.NR ≠ source.unack
    begin source.buffer.free-until (received.NR);
        source.unack := received.NR;
    end
end;
```

validate (received:frame-type; status: status-type);

{ Ez az eljárás ellenőrzi a fogadott keret *address*, *control* és *info* mezőit és ennek megfelelően beállítja a *status* változó értékét }

A következő *send* eljárások előkészítik a megfelelő kereteket a küldéshez. A keretek képzése és küldése a HDLC *frame* struktúráknak megfelelő. A *send-supervisory*, a *send-unnambered* és a *send-CMDR* eljárások törzsei hasonlóan konstruálhatók meg.

```
send-info (VS, VR: sequence-count;
    info-to-send: info-type);
    begin
        INITIATE (transmit, frame) ahol
            frame.address := ... ,
            frame.kind := I,
            frame.pfbit := PF-control.bit,
            frame.NS := VS,
            frame.NR := VR,
            frame.info := info-to-send
    end;

send-supervisory (send-kind : (RR, RNR, REJ);
    VR: sequence-count);

send-unnumbered (send-kind: (SNRM, SARM, DISC, UA));

send-CMDR (control-field: control-type;
    VS, VR: sequence-count; satus: status-type);
```

Távoli akciókezdeményezések

```
transmit (frame);
    { Az átviteli közegtől függően, az alábbiak következhetnek be }

    begin
    case reception of
        loss : ;

    { Az üzenet elveszett az átviteli vonalon }

        FCS-error : ;

    { Átviteli hiba, a keret törlődik }

        CorrectFCS : begin
            received := frame;
            validate (received, status)
        end

    { A keret fogadása hibátlan: a keret érvényesítés
     a megfelelő komponenshez továbbítódik }

    end;
```

5 Ellenőrzőpont komponens

Változók	Jelentés
VS : sequence-count	A *Küldő VS* változójának az értékét tárolja, ha a *P* ill. az $F = 1$
inhibit : boolean	*false*-ra állítódik, ha *P* vagy $F = 1$ kerül küldésre és *true* értéket kap, ha *REJ* fogadására kerül sor és a *P* ill. $F = 0$

Eljárások

setcheckpoint;

{ Ez az eljárás feljegyzi a *Küldő VS* változójának tartalmát ha P/F ciklus kezdődik, azaz ha $P/F = 1$ kerül küldésre }

```
begin
    checkpoint.VS := source.VS;
    inhibit := false;
end;
```

checkpointing;

{ Ez az eljárás kezdeményezheti az *I* keretek újraküldését a P/F bitre épülő hibajavító algoritmus szerint }

```
begin
    if not inhibit ∧
        (received.NR - source.unack) mod modulus ≤
        (checkpoint.VS - source.unack) mod modulus
```

{ Nem minden a *P* vagy $F = 1$ elküldéséig elküldött *I* keret kapott nyugtát a rákövetkező *F* ill. $P = 1$ fogadásáig }

```
    then source.VS := received.NR;
```

{ Újraküldés }

```
end;
```

Megjegyzés: A fenti eljárások a következő esetekben egyszerűsíthető:

(a) -Egyidejűleg mindkét irányú (*FDX*) fizikai átviteli csatorna esetén vagy *ARM* üzemmódban a fenti eljárásokat kell alkalmazni.

(b) Váltakozó egyirányú (*FDX*) vagy *ARM* üzemmódban az *ellenőrzőpont-beállítás* nem igényel akciót és az *ellenőrzés* az *source.VS := received.NR* akcióra redukálódhat.

(c) Ha a *REJ* opció nem kerül alkalmazásra, akkor az *inhibit* értéke mindíg *false*.

6 Kapcsolatkezelő komponens

6.1 Elődleges állomás

Kezdeti állapot
token a *Szétkapcsolt* helyzetben

Átmenet	Feltétel	Akció
SXRM	*PF-control.bit = 1*	*send-unnambered(SXRM);*
	SXRM: SNRM v. SARM az átviteli módtól függően	
UA	*received.kind=UA*	*init(source);* *init(sink);* *init(transmission);*
	Inicializálja a *Küldőt* a *Fogadót* és az *Átvitelt*	
DISC	*PF-control.bit = 1*	*send-unnumbered(DISC);*
CMDR	*received.kind=CMDR*	*init(transmission);*
ERROR	*status in* *[* *invalid-control-field,* *invalid-info,* *invalid-size,* *invalid-NR* *]*	*init(transmission);*
	A beérkezett keret hibát jelöl, amit az *Elsődleges* oldalon magasabb szintű protokollnak kell kezelni	

6.2 Másodlagos állomás

Kezdeti állapot
token a *Szétkapcsolt* helyzetben

Átmenet	Feltétel	Akció
SXRM	*received.kind=SXRM*	*init(transmission);*
	SXRM: SNRM v. SARM az átviteli módtól függően	
UA	*PF-control.bit = 1*	*send-unnumbered(UA);*
DISC	*received.kind=DISC*	*init(transmission);*
CMDR	*true*	*send-CMDR;* *(* *received.control-filed,* *source.VS,sink.VR,* *status* *);*
	Jelenti a protokoll hibát, amelyet az *Elsődleges* oldalon magasabb szintű protokoll kezel	
ERROR	*status in* *[* *invalid-control-field,* *invalid-info,* *invalid-size,* *invalid-NR* *]*	*init(transmission);*
	A beérkezett keret hibát jelöl, amit az *Elsődleges* oldalon magasabb szintű protokollnak kell kezelni	
NSD	*not (received.kind in* *[SNRM, SARM, DISC])*	*init(transmission);*

7 Küldő komponens

Változók	Jelentés
unack : sequence-count	a legrégebbi sorrendben érkezett keret azonosítója, amely érkezése még nem lett nyugtázva

Függvények	
buffer.data (VS: sequence-count);	függvény, amely kiveszi a következő küldendő adatot a *buffer*ből
buffer.to-send (VS: sequence-count);	logkai függvény, amely igaz értéket ad, ha a *buffer*ban van küldendő adat
buffer.free-until (NR: sequence-count);	eljárás, amely felszabadítja a nyugtát kapott adatok *buffer*területét

Kezdeti állapot

token a *Partner vételkész* helyzetben;
VS = 0;
unack = 0;

Átmenet	Feltétel	Akció
$\underline{I}$	*buffer.to-send (VS)* $\wedge$ $VS \neq (unack + window)$ *mod modulus;*	*if PF-control.bit = 1 then* *setcheckpoint*; *send-info (VS,sink.VR,* *buffer.data);* *VS:=VS+1 mod modulus;*
	Ha van küldendő *I* keret, amely belefér a küldő ablakba, akkor a küldésre sor kerülhet	
RNR	*received.kind = RNR*	*examine-NR;* *if received.PF-bit = 1 then* *checkpointing;* *init (transmission);*
RR	*received.kind = RR*	*examine-NR;* *if received.PF-bit = 1 then* *checkpointing;* *init (transmission);*
REJ	*received.kind = REJ*	*examine-NR;* *if received.PF-bit = 0 then* *checkpoint.inhibit := true;* *VS := received.NR;* *init (transmission);*
	A *visszatasítás* akciónak le kell tiltania a P/F checkpoint képzését mindíg újraküldést idéz elő	

8 Fogadó komponens

Változók	Jelentés
VR: sequence-count;	a következő, fogadásra kerülő I keret azonosítója
Függvények	
buffer.space	logikai függvény, amely igaz értéket ad, ha a *buffer*ben van szabad terület adatfogadáshoz
buffer.put (data: info-type);	eljárás, amely adatot továbbít a felhasználónak

Kezdeti állapot

token a *Normál sorrend* helyzetben;

$VR = 0$;

Átmenet	Feltétel	Akció
I	*received.kind* $= I$	*examine-NR;* *if received.pfbit = 1 then* *checkpointing;* *if received.NR = VS then* *forward-data;* *init (transmission)*
	Ha az I keret sorrendben érkezett, továbítja a felhasználó felé a *buffer*en keresztül (ha lehet)	
$\underline{RR}$	*buffer.space*	*if PF-control.bit = 1 then* *setcheckpoint;* *send-supervisory (RR,* *source.VS, VR);*
$\underline{RNR}$	$\neg$*buffer.space*	*if PF-control.bit = 1 then* *setcheckpoint;* *send-supervisory (RR,* *source.VS, VR);*
$\underline{REJ}$	*buffer.space*	*if PF-control.bit = 1 then* *setcheckpoint;* *send-supervisory (RR,* *source.VS, VR);*
$I_{\neq}$	*received.kind* $= I$ $\wedge$ *received.NS*$\neq$*VR*	*examine-NR;* *received.pfbit = 1 then* *checkpointing;* *init (transmission);*
	Az *info* mező figyelmen kívül marad	
$I_{=}$	*received.kind* $= I$ $\wedge$ *received.NS*$=$*VR*	*examine-NR;* *received.pfbit = 1 then* *checkpointing;* *init (transmission);* *forward-data*
	Az adat továbbítódik a felhasználó felé a *buffer*en kersztül (ha lehet)	

```
forward-data();
begin
  if buffer.space then
  begin buffer.put (received.data); VR = VR + 1 mod modulus; end
end;
```

9 PF-kontroll komponens

9.1 Elsődleges állámás

Változók	Jelentés
bit: 0..1;	P/F bit. Az értékét magasabb szintű protokoll réteg állítja dinamikusan

Kezdeti állapot
token a *Nem kérdező* helyzetben

Átmenet	Feltétel	Akció
$\underline{P_0}$	*bit = 0*	
$\underline{P_1}$	*bit = 1*	$timer = t_0$
	Elindítja a *timer*t	
$\underline{P_{1-ism}}$	*status = [timeout]* $\wedge$ *bit = 1*	$timer = t_0$
F_0	*received.pfbit = 0*	*if timer > 0 then* $timer = t_0;$
	Ha a *timer* működik, újraindítja	
F_1	*received.pfbit = 1*	*timer = 0;*
	Megállítja a *timer*t	

9.2 Másodlagos állomás

Változók	Jelentés
bit: 0..1;	P/F bit. Az értékét magasabb szintű protokoll réteg állítja dinamikusan

Kezdeti állapot
token a *Nem kérdezett* helyzetben

Átmenet	Feltétel	Akció
P_0	*received.pfbit = 0*	
P_1	*received.pfbit = 1*	
$\underline{F_0}$	*bit = 0*	
$\underline{F_1}$	*bit = 1*	

10 Óra komponens

Változók	Jelentés
timer : integer;	Időszámláló

Kezdeti állapot
token a *helycsúcsban*

Átmenet	Feltétel	Akció
TICK	*true*	*timer := timer - 1;* *if timer = 0 then* *status := [timeout];*