2. DASAR TEORI

2.1. Cahaya

Cahaya merupakan energi berbentuk gelombang yang membantu manusia untuk dapat melihat benda-benda yang ada di sekitarnya. Kecepatan rambat cahaya adalah 3.10^8 m/detik. Cahaya mempunyai sifat bergerak lurus ke semua arah. Salah satu sumber cahaya yang umum dijumpai adalah cahaya matahari. Cahaya dapat menembus benda yang bening, dibiaskan dan dapat juga dipantulkan ("Cahaya-Wikipedia Indonesia, ensiklopedia bebas berbahasa Indonesia").

Ada beberapa teori yang menjelaskan mengenai fenomena cahaya, seperti teori partikel, teori gelombang, teori elektromagnetik, teori kuantum, dan lain-lainnya. Pada teori partikel dinyatakan bahwa cahaya terdiri dari partikel halus yang memancar ke semua arah dari sumbernya (Isaac Newton, 1675). Pada teori gelombang dinyatakan bahwa cahaya dipancarkan ke semua arah sebagai gelombang. Selain itu juga dinyatakan bahwa gelombang cahaya akan berinterferensi dengan gelombang cahaya yang lain seperti gelombang bunyi, dan cahaya dapat dipolarisasikan (Christiaan Huygens). Pada teori elektromagnetik dikatakan bahwa cahaya adalah gelombang elektromagnetik sehingga tidak memerlukan medium untuk merambat. Selain itu juga dinyatakan bahwa sinar yang dapat dilihat oleh mata adalah bagian dari spektrum elektromagnetik (James Clerk Maxwell). Teori kuantum menyatakan bahwa sinar cahaya terdiri dari paket (kuantum) tenaga yang dikenal sebagai *photon* (Max Planck).

Cahaya mempunyai frekuensi, panjang gelombang, dan kecepatan rambat (Gene Smith, 1999) yang didefinisikan dalam rumus berikut :

$$c = \acute{\imath} \cdot \ddot{e} \qquad (2.1)$$

di mana:

c = kecepatan cahaya (3.10^8 m/detik)

í = frekuensi (Hz)

ë = panjang gelombang

Selain itu cahaya juga mempunyai energi (Gene Smith, 1999) yang dapat dirumuskan sebagai berikut :

$$E = h \cdot í \qquad (2.2)$$

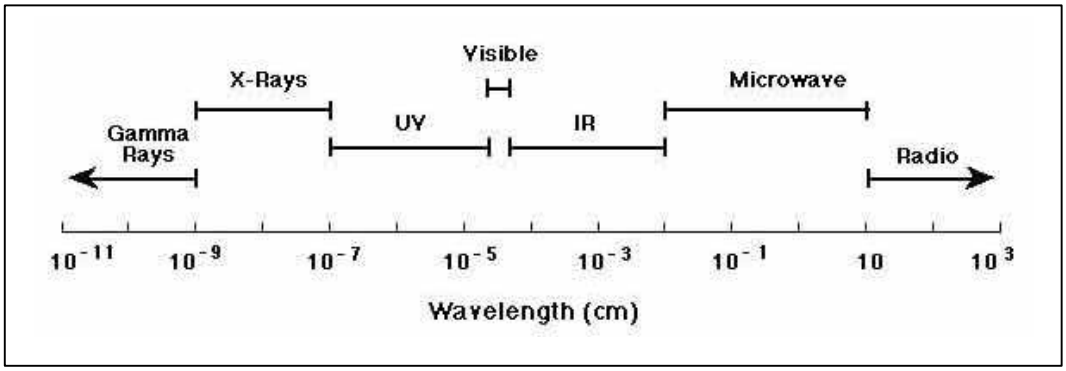
di mana:

E = energi kuantum

h = konstanta Planck (6.625×10^{-27} erg.sec)

í = frekuensi (Hz)

Cahaya tampak, gelombang radio, gelombang mikro, dan *x-ray* merupakan radiasi elektromagnetik yang mempunyai energi dan panjang gelombang tertentu pada spektrum elektromagnetik (Gene Smith, 1999).



Gambar 2.1. Spektrum Elektromagnetik

Sumber: Smith, Gene. *The Fundamental Forces & Light*. 16 April 1999. figure 9. <http://casswww.ucsd.edu/public/tutorial/Forces.html>

2.2. Infrared

Radiasi *infrared* adalah merupakan radiasi elektromagnetik yang mempunyai panjang gelombang di atas cahaya tampak, namun masih lebih pendek daripada gelombang radio. *Infrared* mempunyai panjang gelombang kurang lebih antara 750 nm – 1 mm. Spektrum *infrared* tersebut dibagi menjadi beberapa kategori disesuaikan dengan penggunaannya (“Infrared-Wikipedia, the free encyclopedia”, 2006), antara lain:

- *Near infrared* (NIR, IR-A DIN)

  Mempunyai panjang gelombang 0,75 – 1,4 µm dan sering dipakai dalam komunikasi *fiber optic*.
- *Short wavelength* IR (SWIR, IR-B DIN)

  Mempunyai panjang gelombang 1,4 µm – 3 µm. Panjang gelombang 1530 sampai 1560 nm biasanya digunakan untuk komunikasi jarak jauh.
- *Mid wavelength* IR (MWIR, IR-B DIN)

  Mempunyai panjang gelombang 3 – 8 µm.
- *Long wavelength* IR (LWIR, IR-C DIN)

  Dengan panjang gelombang 8-15 µm.
- *Far infrared* (FIR)

  Dengan panjang gelombang 15 – 1000 µm.

Selain pembagian kategori berdasarkan panjang gelombang seperti di atas, penggunaan *infrared* juga dapat dikategorikan berdasarkan respons dari berbagai macam detektornya (“Infrared-Wikipedia, the free encyclopedia” ,2006), antara lain :

- *Near Infrared* (NIR)

  Dengan panjang 0,7 – 1,0 µm. Area ini direspon oleh *silicon*.
- *Short-wave infrared* (SWIR)

  Dengan panjang 1,0 – 3,0 µm. Area ini direspon oleh salah satunya InGaAs pada panjang gelombang 1,8 µm.
- *Mid-wave infrared* (MWIR)

  Dengan panjang 3 – 5 µm. Area ini direspon oleh InSb HgCdTe dan sebagian PbSe.
- *Long-wave infrared* (LWIR)

  Dengan panjang 8 – 12, atau 7 – 14 µm. Area ini direspon oleh HgCdTe dan *microbolometer*.
- *Very-long wave infrared* (VLWIR)

  Dengan panjang 12 – 30 µm. Area ini direspon oleh *doped silicon*.

Cahaya dan gelombang elektromagnetik pada setiap frekuensi akan memanaskan tiap permukaan yang menyerapnya. Begitu pula *infrared* yang

termasuk dalam bagian kecil spektrum cahaya (“Infrared-Wikipedia, the free encyclopedia”, 2006).

Infrared dimanfaatkan dalam berbagai macam hal, contohnya digunakan dalam perlengkapan *night-vision*, untuk keperluan termografi, untuk *remote* kontrol, untuk perangkat komunikasi *embedded* seperti PDA (*Personnal Digital Assistant*) dan *handphone*, dan lain-lain sebagainya (“Infrared-Wikipedia, the free encyclopedia”, 2006).

2.2.1. Modulasi *infrared*

Untuk mentransmisikan *infrared* sebagai sarana komunikasi, diperlukan adanya modulasi. Dengan menggunakan modulasi, sumber *infrared* akan mengirimkan sinyalnya pada frekuensi tertentu. Kemudian *receiver* yang dapat menerimanya harus di-*tune* pada frekuensi tersebut, sehingga frekuensi yang lainnya akan diabaikan. Dengan cara demikian, *infrared* dapat dimanfaatkan untuk berbagai macam komunikasi dalam suatu tempat tanpa saling menginterferensi (Sam Bergmans, 2005).

Pada *remote* kontrol *infrared*, modulasi biasanya menggunakan frekuensi *carrier* sebesar 32 – 40 KHz. Frekuensi ini dipilih untuk menghindari inteferensi dengan sumber cahaya lainnya seperti matahari atau lampu. Dengan demikian pengiriman sinyal *infrared* oleh *transmitter* dapat diterima dengan baik oleh *receiver*. Sinyal modulasi ini kemudian dikirim ke LED (*Light Emitting Diode*) *infrared* untuk ditransmisikan (“Infrared remote control technology-ePanorama”, 2006). Sistem modulasi yang biasa digunakan dalam pentransmisian *infrared* ini biasanya seperti PAN (*Pulse Amplitude Modulation*), PDM (*Pulse Distance Modulation*), PPM (*Pulse Position Modulation*), dan PCM (*Pulse Code Modulation*).

2.2.2. Protokol *Remote* Kontrol *Infrared*

Pengontrolan suatu alat elektronis secara *wireless* pada era modern ini merupakan hal yang sudah lazim untuk ditemui. Pengontrolan secara *wireless* ini akan memudahkan pengguna dalam melakukan pengontrolan terhadap perangkat elektronis tersebut dari jarak yang cukup jauh. *Remote* kontrol *infrared* adalah

salah satu perangkat kontrol yang paling banyak ditemukan pada perangkat elektronik. Agar tiap *remote* kontrol *infrared* tersebut tidak saling menginteferensi satu perangkat elektronik dengan perangkat elektronik lainnya, diperlukan adanya protokol yang mendefinisikan bagaimana cara menerima data dari sinyal modulasi *infrared remote* tersebut. Beberapa contoh protokol yang dapat ditemukan, contohnya SIRC yang digunakan oleh SONY (San Bergmans, 2003). Kemudian Philips mempunyai beberapa protokol seperti RC-5, RC-6, RC-MM (San Bergmans, 2005). Protokol-protokol ini biasanya menggunakan frekuensi *carrier* yang berbeda. SONY misalnya biasa menggunakan frekuensi 40 KHz, Philips biasa menggunakan frekuensi 36 KHz, dan lain-lain sebagainya. Selain menggunakan frekuensi *carrier* yang berbeda, data yang dikirim tiap *remote* kontrol *infrared* juga dapat berbeda panjangnya. Kemudian faktor lainnya dalam protokol ini adalah sistem *encoding*-nya. Sistem *encoding* ini akan menentukan bagian mana dari sinyal modulasi yang merupakan logik ‘1’ atau logik ‘0’. Logik ‘1’ dan ‘0’ ini akan menentukan data yang ingin dikirimkan oleh *remote* kontrol.

Karena tiap-tiap peralatan elektronik mempunyai protokol yang berbeda-beda, maka diperlukan banyak *remote* kontrol untuk mengontrol masing-masing perangkat elektronik tersebut. Karena itu kemudian muncul perangkat yang dinamakan *Universal Remote Control*. Perangkat ini dapat menggantikan peran dari beberapa *remote* kontrol tersebut karena dapat mengirimkan sinyal *infrared* dengan frekuensi *carrier* yang tidak hanya satu macam.

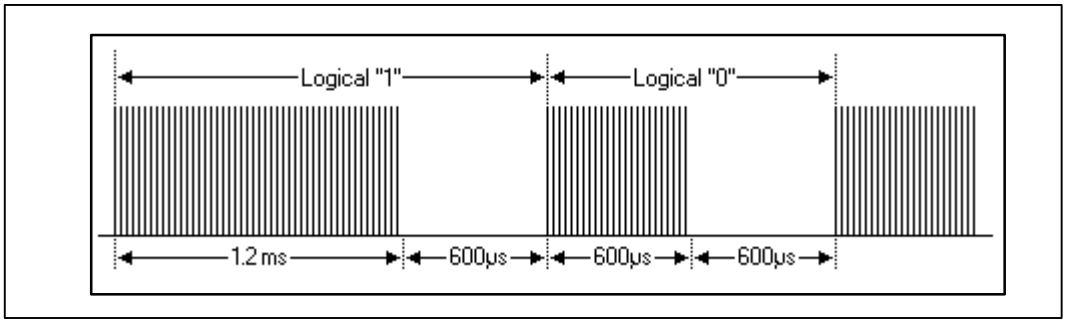
2.2.3. *Encoding Remote* Kontrol *Infrared*

Data *remote* kontrol *infrared* dikirim melalui sinyal modulasi, di mana secara fisik komunikasinya dilakukan oleh sebuah diode *infrared* yang berfungsi sebagai *transmitter* dan *photodiode* sebagai *receiver*. Pada umumnya informasi dikirim dengan tiga macam sistem *encoding* (“SC546 Project An analysis of IR signals used by a remote control”), yaitu :

- *Pulse-coded signal*
- *Space-coded signal*
- *Shift-coded signal*

2.2.3.1. *Pulse-Coded Signal*

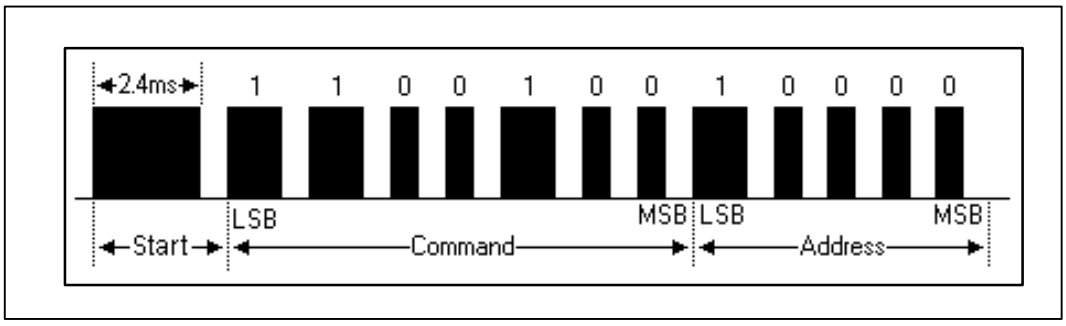
Sistem ini umumnya dimiliki oleh *remote* kontrol merk SONY. Pada sistem ini pulsa mempunyai panjang yang bervariasi untuk merepresentasikan data *remote* kontrol yang dikirimkan. Untuk pengiriman datanya sistem ini menggunakan *Pulse Width Modulation* dengan frekuensi *carrier* 40 KHz. Sistem modulasinya dapat dilihat sebagai berikut :



Gambar 2.2. *Pulse Width Modulation*

Sumber: Bergmans, San. *SB-Projects IR remote control SIRC protocol*. 2006. figure 1. <http://www.sbprojects.com/knowledge/ir/sirc.htm>

Panjang pulsa yang merepresentasikan logika ‘1’ mempunyai panjang 1,2 ms, sedangkan pulsa yang merepresentasikan logika ‘0’ mempunyai panjang 0,6 ms. Tiap logika dipisahkan periode sepanjang 0,6 ms. Panjang data yang dikirimkan ada 3 macam, 12 *bit*, 15 *bit*, dan 20 *bit*. Untuk panjang data yang 12 *bit*, biasanya 5 *bit* dipakai untuk merepresentasikan *address* dan 7 *bit* dipakai untuk merepresentasikan *command*. Untuk panjang data 12 *bit*, struktur data yang dikirim mempunyai spesifikasi sebagai berikut :



Gambar 2.3. *Pulse-Coded Signal*

Sumber: Bergmans, San. *SB-Projects IR remote control SIRC protocol*. 2006. figure 2. <http://www.sbprojects.com/knowledge/ir/sirc.htm>

Data yang dikirimkan selalu didahului dengan *header* logika 1 sepanjang 2,4 ms diikuti dengan selang sepanjang 0,6 ms. Baru kemudian 7 *bit command* dikirimkan diikuti dengan 5 *bit address*. Pada sistem ini LSB (*Least Significant Bit*) dikirimkan terlebih dahulu. Sehingga pada contoh di atas, data *command* yang dikirimkan adalah 13_H, dan *address* 1_H. Data *address* merepresentasikan *device* yang dikontrol oleh *remote*, dan data *command* merepresentasikan tombol *remote* yang ditekan.

Tabel 2.1. Representasi *Bit Address Remote* SONY

Address	*Device*
$01	TV
$02	VCR 1
$03	VCR 2
$06	*Laser Disc Unit*
$0C	*Surround Sound*
$10	*Cassette deck / Tuner*
$11	*CD Player*
$12	*Equalizer*

Sumber: Bergmans, San. *SB-Projects IR remote control SIRC protocol*. 2006. tabel 1.<http://www.sbprojects.com/knowledge/ir/sirc.htm> (telah diolah kembali)

Tabel 2.2. Representasi *Bit Command Remote* Sony

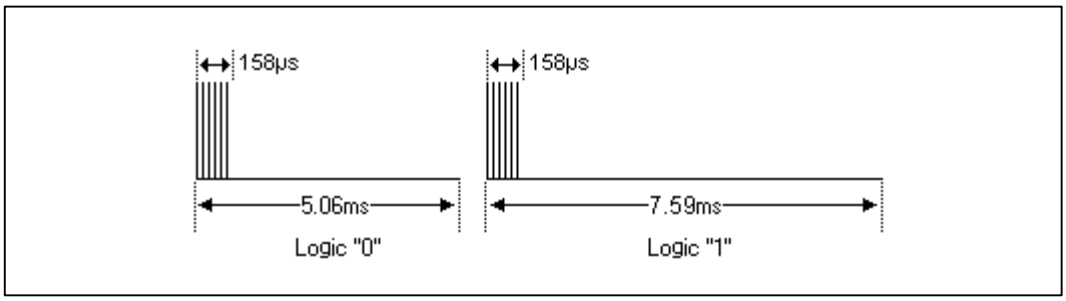
Command	*Function*
$00	*Digit key* 1
$01	*Digit key* 2
$02	*Digit key* 3
$03	*Digit key* 4
$04	*Digit key* 5
$05	*Digit key* 6
$06	*Digit key* 7
$07	*Digit key* 8
$08	*Digit key* 9
$09	*Digit key* 0
$10	*Channel +*
$11	*Channel -*
$12	*Volume +*
$13	*Volume -*

Command	*Function*
$14	*Mute*
$15	*Power*
$16	*Reset*
$17	*Audio Mode*
$18	*Contrast +*
$19	*Contrast -*
$1A	*Colour +*
$1B	*Colour -*
$1E	*Brightness +*
$1F	*Brightness -*
$26	*Balance Left*
$27	*Balance Right*
$2F	*Standby*

Sumber: Bergmans, San. *SB-Projects IR remote control SIRC protocol*. 2006. tabel 2.<http://www.sbprojects.com/knowledge/ir/sirc.htm> (telah diolah kembali)

2.2.3.2. *Space-Coded Signal*

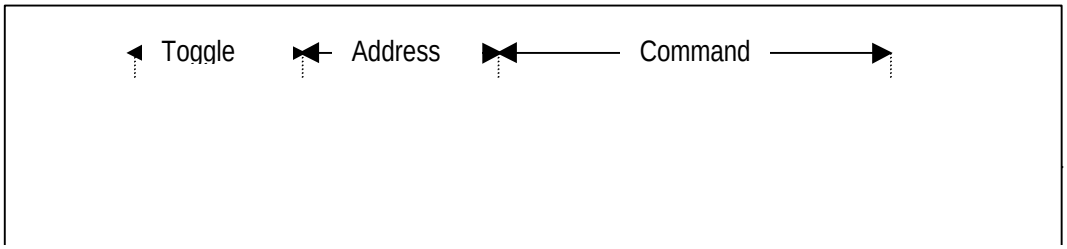
Sistem ini dimiliki oleh *remote* kontrol Panasonic dan dikenal dengan nama REC-80. Pada sistem ini selang waktu antara pulsa merepresentasikan data yang dikirim. Untuk mengirimkan datanya sistem ini menggunakan *Pulse Distance Modulation* dengan frekuensi *carrier* 38 KHz. Sistem modulasinya dapat dilihat sebagai berikut :

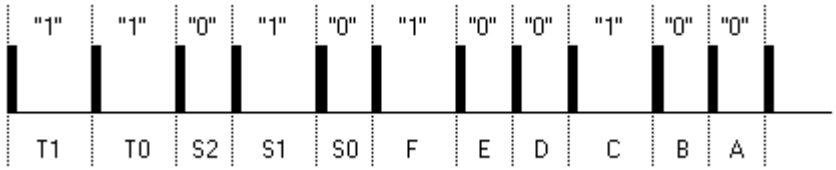


Gambar 2.4. *Pulse Distance Modulation*

Sumber: Bergmans, San. *SB-Projects IR remote control Philips RECS-80*. 2006. figure 1. <http://www.sbprojects.com/knowledge/ir/recs80.htm>

Logika ‘0’ ditandai dengan *carrier* sepanjang 158 µs dilanjutkan dengan *space*, dimana panjang totalnya adalah 5,06ms. Logika ‘1’ ditandai dengan *carrier* sepanjang 158 µs dilanjutkan dengan *space* yang panjang totalnya adalah 7,59ms. Struktur data yang dikirim dapat dilihat sebagai berikut :





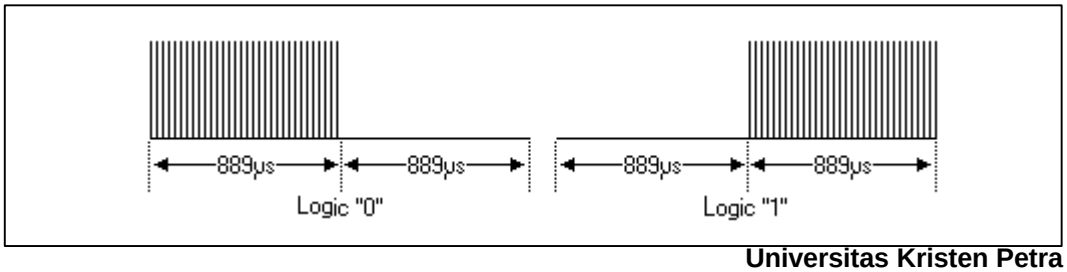
Gambar 2.5. *Space-Coded Signal*

Sumber: Bergmans, San. *SB-Projects IR remote control Philips RECS-80*. 2006. figure 2. <http://www.sbprojects.com/knowledge/ir/recs80.htm> (telah diolah kembali)

Contoh pada gambar 2.5 mengirimkan *command* 36_D ke *address* 4_D. Biasanya pulsa pertama adalah merupakan pulsa referensi, dengan nilai '1'. *Receiver* dapat menggunakan *bit* ini untuk menghitung panjang sinyal tiap *bit*. *Bit* berikutnya adalah merupakan *bit toggle*, dimana nilainya selalu berubah apabila tombol *remote* dilepas. *Bit* ini dapat digunakan *receiver* untuk membedakan tombol baru yang ditekan atau tombol yang sama yang ditahan oleh pengguna. Tiga pulsa selanjutnya (S2, S1, S0), merepresentasikan *bit address device* yang dikontrol oleh *remote* tersebut, dimana MSB (*Most Significant Bit*) dikirim terlebih dulu. Pulsa F sampai A merepresentasikan *command remote* yang dikirim. Jika tombol *remote* ditahan, maka pulsa *command* akan dikirim secara terus menerus dengan *rating* 121,5ms.

2.2.3.3. *Shift-Coded Signal*

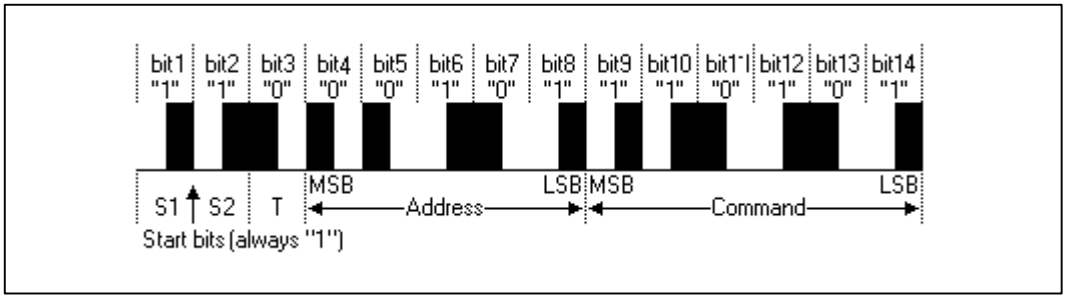
Sistem ini dimiliki oleh *remote* kontrol Philips dan dikenal dengan nama RC-5. Pada sistem ini, transisi tiap *bit* yang menentukan data yang dikirim. Apabila transisi naik, maka data yang dikirim adalah logika '1'. Dan bila transisi turun, maka data yang dikirim adalah logika '0'. Untuk mengirimkan datanya, sistem ini menggunakan modulasi *bi-phase* atau lebih dikenal dengan *coding* Manchester, dengan frekuensi *carrier* 36 KHz. Sistem modulasinya dapat dilihat sebagai berikut :



Gambar 2.6. *Bi-Phase Modulation*

Sumber: Bergmans, San. *SB-Projects IR remote control Philips RC-5*. 2006. figure 1. <http://www.sbprojects.com/knowledge/ir/rc5.htm>

Pada modulasi ini setiap *bit* mempunyai panjang pulsa yang sama (1,778 ms), dimana setengah dari panjang pulsa tersebut diisi dengan frekuensi *carrier*. Struktur data yang dikirimkan dapat dilihat sebagai berikut :



Gambar 2.7. *Shift-Coded Signal*

Sumber: Bergmans, San. *SB-Projects IR remote control Philips RC-5*. 2006. figure 2. <http://www.sbprojects.com/knowledge/ir/rc5.htm>

Dua pulsa pertama adalah merupakan pulsa *start*, dan keduanya bernilai logik ‘1’. *Bit* ketiga adalah *bit toggle*, yang digunakan untuk membedakan apakah suatu tombol sedang ditahan atau ditekan secara berulang-ulang. Nilai *bit* ini akan berubah apabila sebuah tombol dilepas dan kemudian ditekan lagi. Lima *bit* berikutnya adalah *bit address* yang merepresentasikan *device* yang diakses oleh *remote*. *Bit* yang dikirim terlebih dulu adalah *bit* MSB (*Most Significant Bit*). Lima *bit* terakhir adalah *bit command*, yang merepresentasikan tombol *remote* yang ditekan. Seperti *bit address*, pada *bit command* ini *bit* yang dikirim terlebih dulu adalah MSB.

Tabel 2.3. Representasi *Command* Bit RC-5

RC-5 *Command*	TV *Command*	VCR *Command*
$00	0	0

$01	1	1
$02	2	2
$03	3	3
$04	4	4
$05	5	5
$06	6	6
$07	7	7
$08	8	8
$09	9	9
$0A	-/--	-/--
$0C	*Standby*	*Standby*
$0D	*Mute*	
$10	*Volume +*	
$11	*Volume -*	

Tabel 2.3. Representasi *Command* Bit RC-5 (sambungan)

RC-5 *Command*	TV *Command*	VCR *Command*
$12	*Brightness +*	
$13	*Brightness -*	
$20	*Program +*	*Program +*
$21	*Program -*	*Program -*
$32		*Fast Rewind*
$34		*Fast Forward*
$35		*Play*
$36		*Stop*
$37		*Recording*

Sumber: Bergmans, San. *SB-Projects IR remote control Philips RC-5*. 2006. tabel 1. <http://www.sbprojects.com/knowledge/ir/rc5.htm> (telah diolah kembali)

Tabel 2.4. Representasi *Address* Bit RC-5

RC-5 *Address*	*Device*
$00	TV 1
$01	TV 2
$02	*Teletext*
$03	*Video*
$04	LV1
$05	VCR1
$06	VCR2
$07	*Experimental*
$08	*Sat*1
$09	*Camera*
$0A	*Sat*2
$0B	
$0C	CDV
$0D	*Camcorder*
$0E	

RC-5 *Address*	*Device*
$0F	
$10	*Pre-amp*
$11	*Tuner*
$12	*Recorder*1
$13	*Pre-amp*
$14	CD *Player*
$15	*Phono*
$16	SatA
$17	*Recorder*2
$18	
$19	
$1A	CDR
$1B	
$1C	
$1D	*Lighting*

Sumber: Bergmans, San. *SB-Projects IR remote control Philips RC-5*. 2006. tabel 2. <http://www.sbprojects.com/knowledge/ir/rc5.htm> (telah diolah kembali)

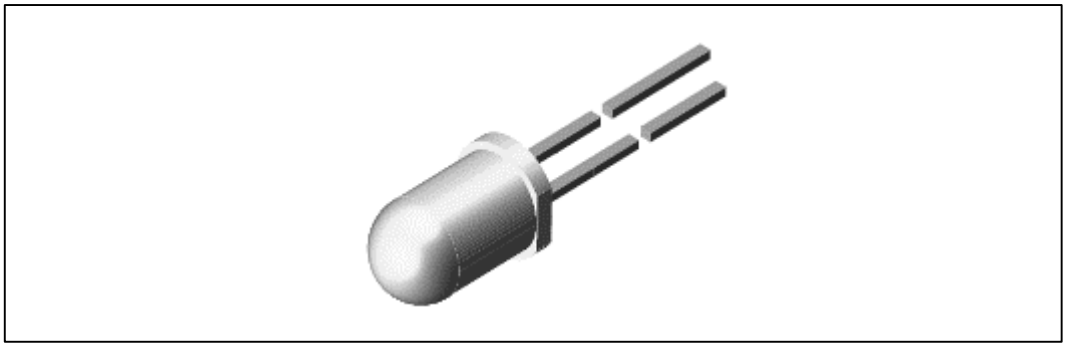
2.3. Hardware

2.3.1. *Transmitter Infrared*

Pada umumnya *remote* kontrol menggunakan transmisi sinyal *infrared* yang dimodulasi dengan sinyal *carrier* dengan frekuensi tertentu, yaitu pada frekuensi 30 KHz sampai 40 KHz. Sinyal ini kemudian dipancarkan melalui LED *Infrared*. Sinyal yang dipancarkan oleh *transmitter* diterima oleh *receiver infrared* dan kemudian di-*decode*-kan sebagai sebuah paket data biner.

Panjang sinyal data biner ini bervariasi antara satu *vendor* dengan *vendor* lainnya sehingga suatu *remote* kontrol hanya dapat digunakan untuk sebuah produk dari *vendor* yang sama dan pada tipe yang sama. Hal ini dapat dicontohkan pada *remote* TV SONY hanya bisa digunakan untuk mengontrol *receiver* TV SONY dan sebaliknya, tetapi tidak dapat digunakan untuk *receiver* TV merek yang lain.

Infrared dapat digunakan baik untuk memancarkan data maupun sinyal lainnya, misalnya sinyal suara. Keduanya membutuhkan sinyal *carrier* untuk membawa sinyal data maupun sinyal suara tersebut hingga sampai pada *receiver*. Untuk transmisi data biasanya sinyal ditransmisikan dalam bentuk pulsa-pulsa. Ketika sebuah tombol ditekan pada *remote* kontrol, maka *remote* kontrol *infrared* akan mentransmisikan sinyal yang akan dideteksi *receiver* sebagai urutan data biner.



Gambar 2.8. *Infrared Emmitting Diode*

Sumber: Vishay Semiconductor. *TSAL6200 Datasheet*. 2003. p.1. <http://www.innovativeelectronics.com/innovative_electronics/download_files/datasheet/TSAL6200.pdf>

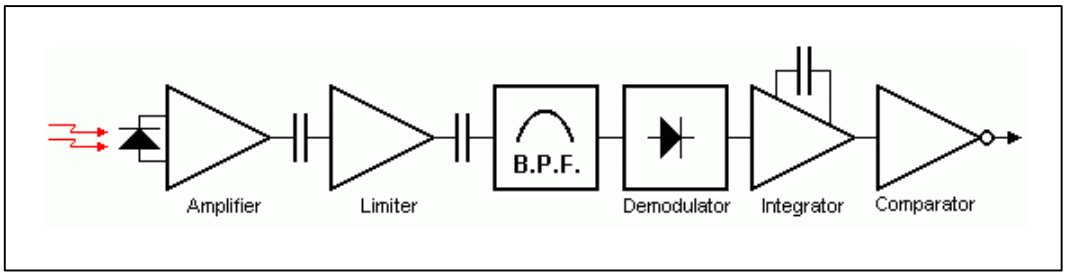
2.3.2. *Receiver infrared*

Komponen yang dapat menerima *infrared* ini merupakan komponen yang peka cahaya yang dapat berupa *dioda* (*photodioda*) atau *transistor* (*phototransistor*). Komponen ini akan merubah energi cahaya, dalam hal ini energi cahaya *infrared*, menjadi pulsa-pulsa sinyal listrik. Komponen ini harus mampu mengumpulkan sinyal *infrared* sebanyak mungkin sehingga pulsa-pulsa sinyal listrik yang dihasilkan kualitasnya cukup baik. Semakin besar intensitas *infrared* yang diterima, maka sinyal pulsa listrik yang dihasilkan akan baik. Jika

sinyal *infrared* yang diterima intensitasnya lemah, maka *receiver infrared* tersebut harus mempunyai pengumpul cahaya (*light collector*) yang cukup baik dan sinyal pulsa yang dihasilkan oleh sensor *infrared* ini harus dikuatkan.

Pada prakteknya sinyal *infrared* yang diterima intensitasnya sangat kecil sehingga perlu dikuatkan. Selain itu agar tidak terganggu oleh sinyal cahaya lain, maka sinyal listrik yang dihasilkan oleh *receiver infrared* harus di-*filter* pada frekuensi sinyal *carrier* yaitu pada 30 KHz sampai 40 KHz. Selanjutnya baik *photodioda* maupun *phototransistor* disebut sebagai *photodetector*.

Modul *receiver infrared remote* kontrol, biasa menggunakan *photodetector* sebagai mediumnya. Umumnya modul tersebut mengandung beberapa komponen untuk mengolah sinyal *infrared* yang diterimanya. Blok diagram dari suatu modul *receiver infrared* dapat dilihat sebagai berikut :



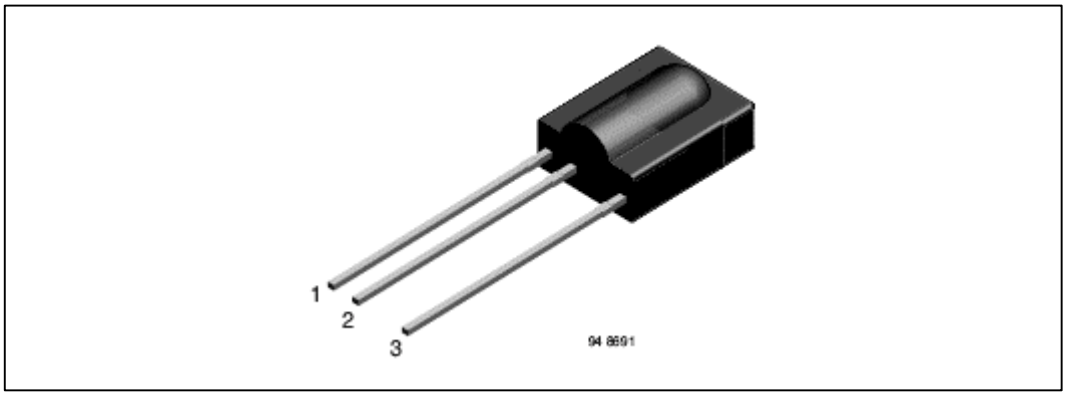
Gambar 2.9. Blok Diagram *Receiver Infrared*

Sumber: Bergmans, San. *SB-Projects IR remote control*. 2006. figure 5 <http://www.sbprojects.com/knowledge/ir/ir.htm>

Sinyal *Infrared* akan diterima oleh *photodetector* dan kemudian akan dikuatkan oleh *amplifier*. Kemudian *limiter* bertugas untuk mendapatkan *level* pulsa sinyal *infrared* secara konstan, dari jarak manapun sinyal diterima. *Band Pass Filter* kemudian bertugas untuk menyaring frekuensi *carrier* yang diterima oleh *receiver* ini. Tiga komponen selanjutnya bertugas untuk mengolah frekuensi modulasi yang diterima (San Bergmans, 2001). Blok diagram pada gambar 2.9 merupakan modul-modul yang sudah *embedded* di dalam satu fisik perangkat pada gambar 2.10. Perangkat tersebut mempunyai tiga *pin* yang masing-masing fungsinya sebagai berikut:

- *Pin* 1 : GND (*ground*)
- *Pin* 2 : VS (*input* tegangan *supply*)

- *Pin* 3 : OUT (*output*)

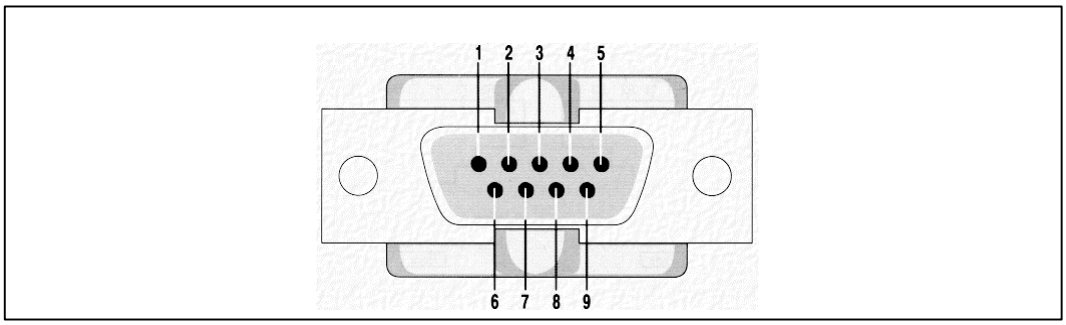


Gambar 2.10. Modul *Receiver Infrared*

Sumber: Vishay Semiconductor. *TSOP12XX Datasheet*. 2003. p.1. <http://tmt.robot.free.fr/2004/balise/docs/TSOP12XX.pdf>

2.3.3. Serial *Port*

Serial *port* merupakan salah satu standar koneksi antara *device* dengan sebuah komputer. Standar koneksi ini merupakan salah satu standar koneksi tertua pada komputer selain paralel *port*. Konektor serial pada PC umumnya merupakan konektor serial bertipe *male* dengan *pin* berjumlah 9 buah. Pada penggunaannya, *pin-pin* tersebut tidak selalu digunakan semua. Serial *port* dapat mengirim dan menerima data satu *bit* dalam suatu waktu melalui satu kabel. Pada kenyataannya serial *port* dapat berkomunikasi dua arah (*full duplex*), dengan menggunakan tiga kabel (*null modem*). Satu kabel digunakan untuk mengirim, satu kabel untuk menerima, dan satu kabel untuk sinyal *ground*. Gambar konektor serial *port* dapat dilihat pada gambar 2.11.



Gambar 2.11. Konektor Serial PC

Sumber: ARC Electronics. *RS-232 Data Interface*. September 2006. figure 5. <http://www.arcelect.com/rs232.htm>

Tabel 2.5. Deskripsi Pin Serial *Port*

	DB-9 Pin	*Name*	Dir	*Description*
←	1	DCD		*Carrier Detect*
←	2	RXD		*Receive Data*
→	3	TXD		*Transmit Data*
→	4	DTR		*Data Terminal Ready*
—	5	GND		*System Ground*
←	6	DSR		*Data Set Ready*
→	7	RTS		*Request to Send*
←	8	CTS		*Clear to Send*
←	9	RI		*Ring Indicator*

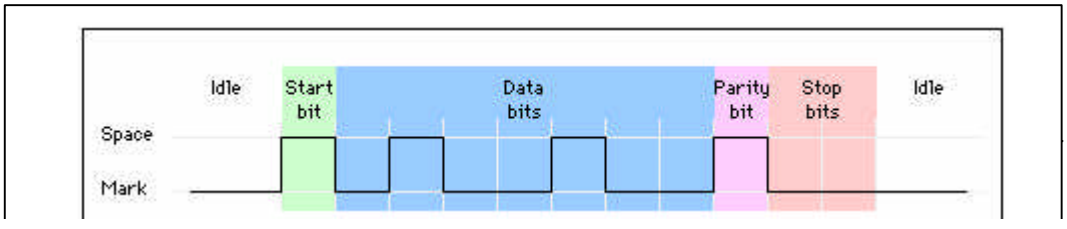
Sumber: PC Control. *Easy View Serial Port Connector Pinout.* 2004. tabel 1 <http://www.pc-control.co.uk/Serial-Pinout.htm> (telah diolah kembali)

Selain itu juga perlu diketahui karakteristik elektris *port* serial sebagai berikut :

- "Space" (logika 0) ialah tegangan antara + 3 hingga +25 V.
- "Mark" (logika 1) ialah tegangan antara –3 hingga –25 V.
- Daerah antara + 3V hingga –3V tidak didefinisikan /tidak terpakai.
- Tegangan *open circuit* tidak boleh melebihi 25 V.
- Arus hubungan singkat tidak boleh melebihi 500mA.

Ada dua macam tipe dasar dalam komunikasi serial, yaitu *synchronous* dan *asynchronous*. Pada komunikasi *synchronous*, dua *device* saling mensinkronisasikan dirinya saat komunikasi awal, dan kemudian secara konstan saling mengirimkan *bit-bit*, baik itu *bit* data yang ingin dikirimkan maupun *bit idle*. Dengan cara ini *device* akan mengetahui bahwa koneksi keduanya masih sedang berlangsung. Komunikasi *asynchronous* memerlukan *start bit* dan *stop bit* dalam komunikasinya, untuk menandakan *byte* data yang ingin dikirim. *Byte* data yang ingin dikirim diawali dengan *start bit* dan diakhiri dengan *stop bit*. *Port* serial pada PC biasanya menggunakan sistem komunikasi *asynchronous*.

Untuk melakukan komunikasi serial, harus ditentukan empat parameter dasar, yaitu *baud rate*, jumlah data *bit*, *parity bit*, dan jumlah *stop bit*. Empat parameter tersebut menyusun *frame* data yang akan dikirim dalam komunikasi serial ("TALtech – Introduction to RS232 Serial Communications").



Gambar 2.12. *Frame* Komunikasi Serial

Sumber: Francis Courtois. *Baud Rate, Data Bits, Parity*. 1997. figure 1. <http://francis.courtois.free.fr/jc1/serial/Basics/BitFormat.html>

Baud Rate adalah tolok ukur kecepatan aliran data antara *device* yang menggunakan komunikasi serial. Komunikasi ini menggunakan dua referensi tegangan untuk melakukan komunikasi, yaitu referensi tegangan yang disebut *Mark* dan *Space*. Referensi tegangan *Mark* bernilai negatif, dan referensi tegangan *Space* bernilai positif. Dengan *coding* seperti di atas, *baud rate* menyatakan jumlah *bit* informasi (termasuk *control bit*) yang dikirim perdetik.

Sinyal *start bit* menandakan dimulainya sebuah *frame* data. *Start bit* ditandai oleh transisi tegangan *Mark* ke tegangan *Space*. Jika digunakan *baud rate* 9600 dalam pentransmisian data ini, maka periode waktu *start bit* adalah 1/9600 = 0,104ms. Sehingga untuk mengirimkan satu *frame*, waktu yang dibutuhkan adalah 1,146ms.

Data bit pada komunikasi serial dikirimkan mulai dari LSB (*Least Significant Bit*) ke MSB (*Most Significant Bit*). Sehingga LSB berada pada *bit* paling kanan, dan MSB berada pada *bit* paling kiri. Untuk membaca data bit ini, tegangan negatif dijadikan sebagai logika ‘1’ dan tegangan positif dijadikan sebagai logika ‘0’.

Parity bit merupakan *bit optional* yang dapat dipakai atau tidak dalam proses komunikasi serial. Logika *parity bit* sama dengan *data bit*, yaitu logika ‘1’ bernilai tegangan negatif dan logika ‘0’ bernilai tegangan positif. Bit ini biasanya digunakan untuk keperluan pengecekan *error* dalam transmisi. Misalnya ditentukan bahwa paritas transmisi yang dilakukan ganjil. Apabila jumlah logika ‘1’ pada data genap, *bit* paritas ini akan di-*set* dengan logika ‘1’. Apabila logika ‘1’ pada data sudah ganjil, *bit* paritas ini di-*set* logika ‘0’.

Stop bit merupakan bagian terakhir dari *frame* data transmisi. *Bit* ini direpresentasikan oleh tegangan negatif. Apabila tidak ada data lagi yang dikirimkan, maka sinyal transmisi berada pada level tegangan negatif ("Baud Rate, Data Bits, Parity", 1997).

2.4. Software

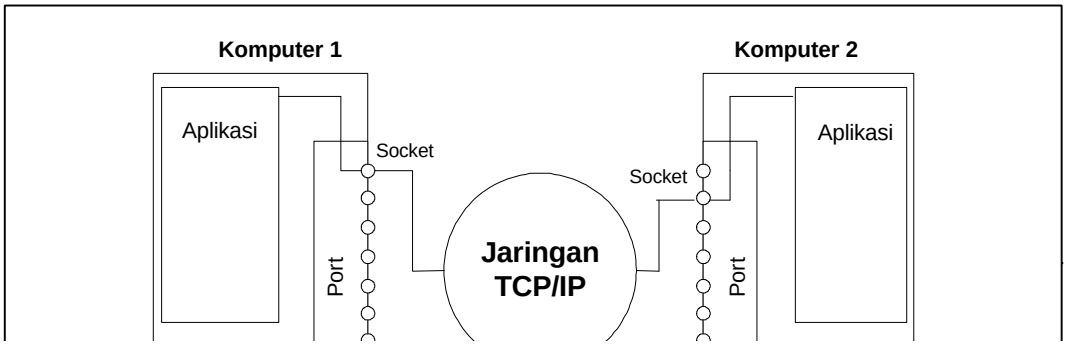
2.4.1. *Socket* dan Protokol

Socket adalah sebuah penghubung antar komputer dalam sebuah jaringan TCP/IP. *Socket* digunakan untuk bertukar berbagai macam informasi seperti *file*, *mail*, *website*, *multiplayer game* dan sebagainya dalam sebuah jaringan komputer.

Ada 2 macam tipe protokol yang digunakan dalam jaringan komputer, TCP (*Transmission Control Protocol*) dan UDP (*User Datagram Protocol*). Protokol adalah seperangkat aturan yang digunakan untuk menentukan dengan cara apa informasi yang ada dikirim dan diterima dalam sebuah jaringan TCP/IP. Protokol-protokol yang lain seringkali digunakan di atas TCP dan UDP, misalnya *File Transfer Protocol* (FTP), sebuah protokol untuk pertukaran *file* yang menggunakan TCP sebagai penghubung dengan *file server*. TCP dan UDP sendiri dibangun di atas IP (*Internet Protocol*).

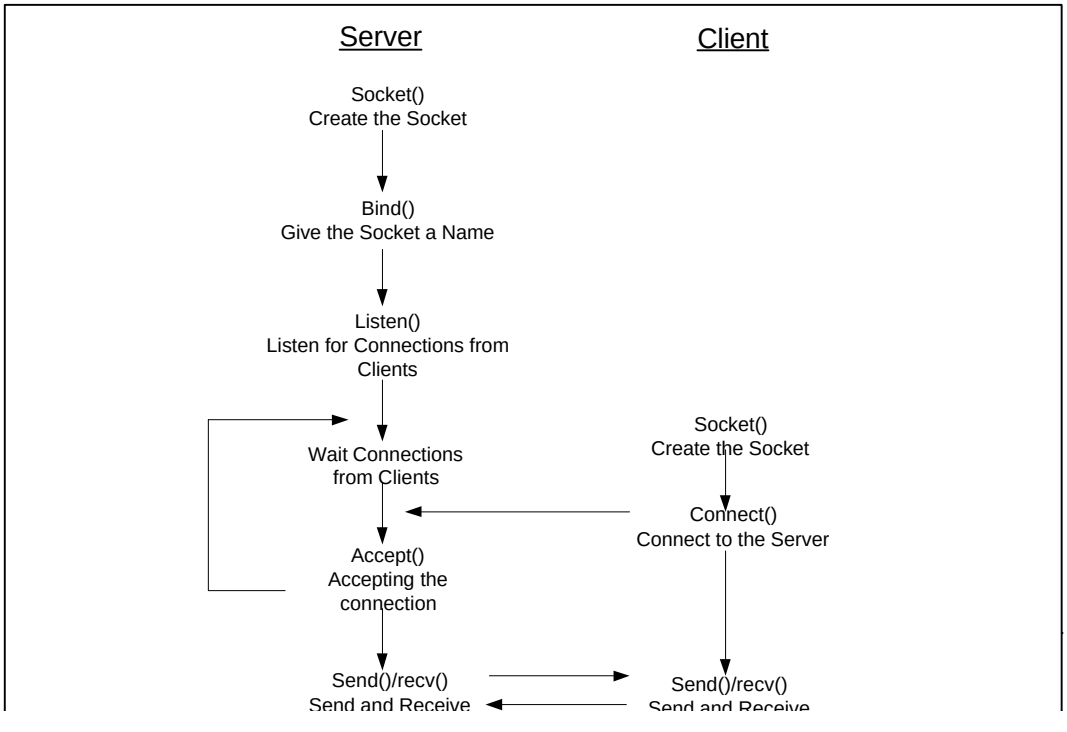
Sebuah *socket* TCP bisa menjamin bahwa informasi yang dikirimkan sampai di tujuan dengan baik, sedangkan *socket* UDP tidak bisa menjamin sampai atau tidaknya informasi yang dikirimkan ke tujuan sebelum pihak penerima memberikan balasan.

Untuk menghubungkan *socket* pada sebuah komputer dengan *socket* pada komputer lainnya, perlu untuk mengetahui IP *address* komputer yang dituju beserta *port* yang akan dituju. Tiap komputer memiliki 65536 *port*. Beberapa *port* biasanya telah digunakan oleh beberapa protokol, misalnya *port* 80 biasanya digunakan untuk HTTP (*Hypertext Transfer Protocol*), *port* 21 digunakan untuk FTP (*File Transfer Protocol*).



Gambar 2.13. Hubungan antar *Socket* pada 2 Komputer

Komunikasi TCP/IP biasanya melibatkan 2 pihak, yaitu *server* dan *client*. Untuk dapat memulai komunikasi TCP/IP, sebuah *server* akan membuat koneksi *socket*, dan kemudian menamai *socket* tersebut. Setelah *socket* terbentuk, *server* akan menunggu koneksi *client*. Saat *client* diterima, *server* akan membuatkan *socket* baru untuk *client* tersebut sebagai jalur komunikasi *server* dengan *client* tersebut, sedangkan *socket* yang pertama kali dibuat digunakan untuk menunggu koneksi dari *client* yang lainnya.



Gambar 2.14. Proses Komunikasi TCP/IP

Sumber: Dumas, Arthur. *Programming Winsock 1st edition*. Indianapolis: SAMS Publishing. 1995. p.46 (telah diolah kembali)

2.4.2. *Winsock*

Winsock (`Winsock 2`) merupakan standar untuk pemrograman jaringan pada sistem operasi Windows. `Winsock 2` diciptakan sebagai antarmuka pemrograman TCP/IP pada semua versi sistem operasi Windows mulai dari Windows 3.x, Windows 95/98, Windows NT, Windows 2000, dan Windows XP.

Fitur-fitur yang dimiliki `Winsock 2`, antara lain :

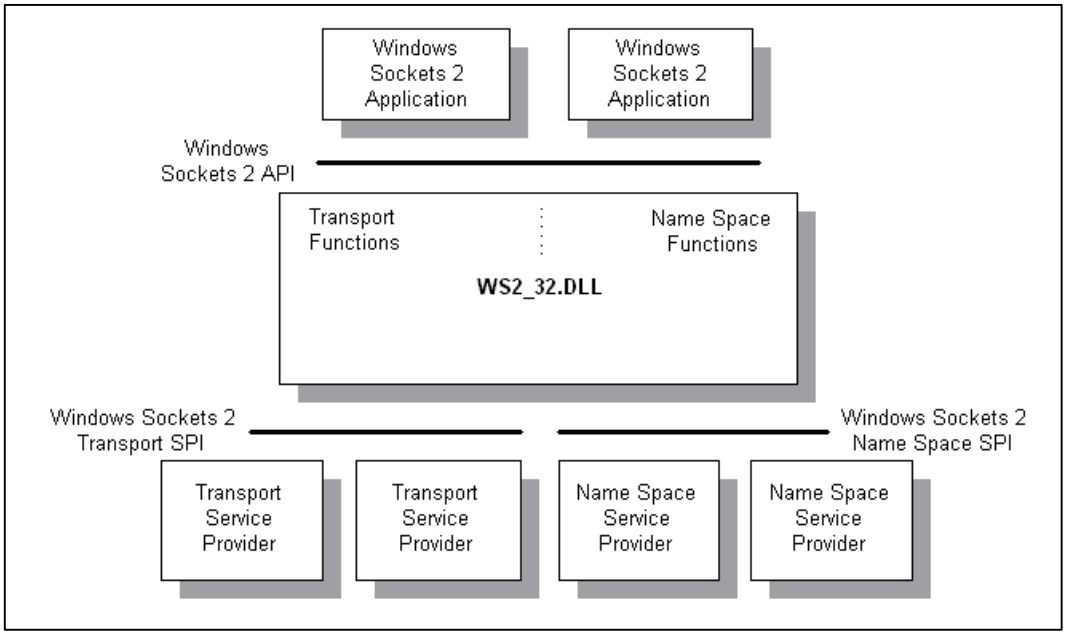
- `Winsock 2` mampu bekerja pada beberapa protokol, tidak hanya pada protokol TCP/IP saja, karena `winsock 2` memiliki fungsi-fungsi baru yang dibutuhkan untuk menangani beberapa protokol. Kemampuan menangani beberapa protokol ini lebih menguntungkan karena pada *winsock* sebelumnya hanya mampu menangani protokol TCP/IP saja.
- `Winsock 2` menyediakan sebuah standar antarmuka yang disebut SPI yang menghubungkan antara DLL *winsock* ("`WS2_32.dll`") dengan *protocol stack*. Dengan demikian, "`WS2_32.dll`" dapat mengakses beberapa *protocol stack* dari beberapa perusahaan yang berbeda secara simultan. Tidak seperti `winsock 1.1` yang mengharuskan tiap-tiap perusahaan pembuat *protocol stack* menyediakan *library* untuk mengimplementasikan antarmuka *winsock*. Tentu saja, *library* yang dibuat oleh suatu perusahaan berbeda dengan *library* yang dibuat oleh perusahaan lainnya, dan ini akan sangat menyulitkan seorang pemrogram bila harus menulis ulang aplikasinya agar dapat berjalan pada protokol yang lain.

- Kode sumber aplikasi yang ditulis dengan `winsock 1.1` dapat berjalan dengan mudah disesuaikan dengan sistem `winsock 2` tanpa melakukan modifikasi terlalu banyak pada kode sumber.

2.4.3. Arsitektur *Winsock*

Arsitektur `winsock 1.1` membatasi hanya satu "`winsock.dll`" yang aktif dalam sebuah sistem. Hal ini menyebabkan, sistem akan kesulitan untuk mengimplementasikan lebih dari satu *winsock* dalam waktu yang bersamaan.

`Winsock 2` memiliki arsitektur yang lebih fleksibel sehingga mampu mendukung *multiple protocol stack*, antarmuka dan *service provider* secara simultan. Pada bagian *layer* atas `winsock 2` memang masih menggunakan satu DLL (`Winsock 2` API), namun terdapat *layer* lagi di bawahnya dan sebuah *service provider interface* standar, yang mana masing-masing juga bersifat fleksibel.



Gambar 2.15. Arsitektur Winsock 2

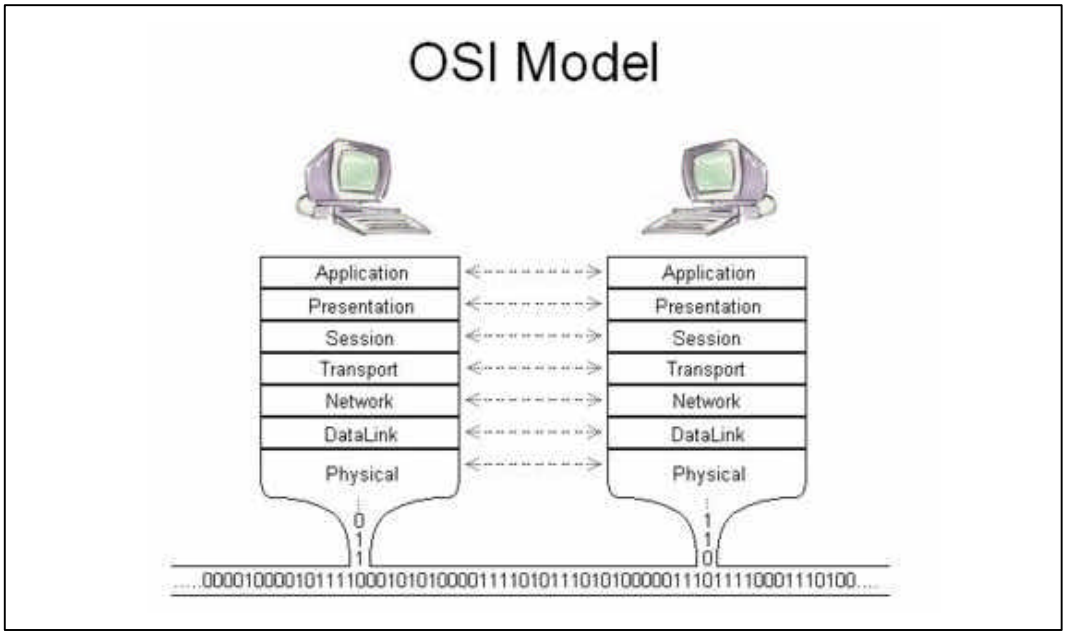
Sumber: Microsoft. MSDN Help. *Winsock Sockets 2 Architecture*. 23 January 2004.<ms-help://MS.MSDNQTR..2002APR.1033/winsock/ovrvw1_5kky.htm>

`Winsock 2` mengadopsi model Windows *Open Systems Architecture* yang memisahkan API dari *Protocol Service Provider*. Dengan menggunakan model ini, *winsock* menyediakan standar API, dan tiap-tiap *vendor* menyediakan *layer service provider*-nya sendiri. Layer API berhubungan dengan layer *service provider* melalui sebuah standar *Service Provider Interface* (SPI), yang mana SPI tersebut mampu menghubungkan *layer* API dengan *layer service provider* dari berbagai *vendor*.

Beberapa fasilitas baru `winsock 2` membutuhkan banyak perubahan pada arsitekturnya. Meskipun arsitektur `winsock 2` jauh berbeda dengan arsitektur `winsock 1.1`, `winsock 2` masih mendukung aplikasi-aplikasi yang ditulis dengan winsock versi sebelumnya. Namun demikian, penanganan "`WS2_32.dll`" untuk aplikasi-aplikasi yang ditulis dengan `winsock 1.1` baik yang 16-*bit* maupun 32-*bit*, memiliki sedikit perbedaan dengan aplikasi yang ditulis dengan `winsock 2`.

2.4.4. OSI (*Open System Interconnection*)

Model OSI (*Open System Interconnection*) dikembangkan sebagai model untuk arsitektur komunikasi komputer, serta sebagai kerangka kerja bagi pengembangan standar-standar protokol. Model OSI menggambarkan bagaimana informasi dari suatu aplikasi pada sebuah komputer berpindah melewati sebuah media jaringan ke suatu aplikasi di komputer lain. Model ini ditujukan bagi pengkoneksian *open system*, karena model OSI ini merupakan sistem yang terbuka untuk melakukan komunikasi dengan sistem-sistem. Model OSI secara konseptual terdiri dari tujuh *layer*, dimana masing-masing *layer* memiliki fungsi yang khusus.



Gambar 2.16. Model OSI

Sumber: Wikipedia, the free encyclopedia. *OSI model*. 2006. figure 1. <http://en.wikipedia.org/wiki/OSI_model>

Tujuh *layer* pada model OSI dapat dibagi menjadi dua bagian, *layer* atas dan *layer* bawah. *Layer* atas berhubungan dengan persoalan aplikasi dan biasanya diimplementasikan pada *software*. *Layer-layer* yang termasuk *layer* atas adalah *application layer*, *presentation layer*, dan *session layer*. Sedangkan *layer* bawah berhubungan dengan persoalan *transport* data. *Layer-layer* yang termasuk *layer* bawah adalah *transport layer*, *network layer*, *data-link layer*, dan *physical layer*. *Physical layer* dan *data-link layer* diimplementasikan dalam *hardware* dan *software*. *Layer-layer* bawah yang lain umumnya hanya diimplementasikan dalam *software*. Berikut ini penjelasan fungsi-fungsi khusus yang ada pada setiap *layer*.

- ***Application layer :*** menyediakan suatu cara bagi program-program aplikasi untuk mengakses lingkungan OSI. *Layer* ini berisi fungsi-fungsi manajemen dan mekanisme-mekanisme yang umumnya berguna untuk mendukung aplikasi-aplikasi yang didistribusikan. Selain itu, aplikasi dengan tujuan penggunaan umum seperti *file transfer*, *email*, dan *terminal access* untuk komputer-komputer yang berjauhan ditempatkan pada *layer* ini.

- ***Presentation layer :*** menentukan format data yang dipindahkan antar aplikasi dan menawarkan pada program-program aplikasi serangkaian layanan transformasi data. *Layer* ini menentukan sintaks yang dipergunakan antar aplikasi serta menyediakan modifikasi seleksi dan *subsequent* dari representasi yang dipergunakan. Contoh layanan khusus pada *layer* ini adalah kompresi dan enkripsi data.
- ***Session layer :*** menyediakan mekanisme untuk mengatur dialog antar aplikasi pada ujung-ujung sistem. Pada beberapa kasus, akan ada sedikit atau bahkan sama sekali tidak diperlukan layanan dari *layer* ini, namun untuk beberapa aplikasi tertentu, layanan pada *layer* ini tetap diperlukan.
- ***Transport layer :*** menyediakan suatu mekanisme perubahan data di antara ujung-ujung sistem. Layanan *transport* pada *connection-oriented* menjamin bahwa data yang dikirim bebas dari kesalahan, secara bertahap, dengan tidak mengalami duplikasi atau hilang. *Layer* ini juga dapat dikaitkan dengan mengoptimalisasikan penggunaan layanan jaringan dan menyediakan mutu layanan yang bisa diminta untuk entiti sesi. Sebagai contoh, entiti sesi bisa menentukan laju *error* yang boleh diterima, maksimum penundaan, prioritas dan pengamanan. Ukuran dan kelengkapan sebuah protokol *transport* tergantung dari seberapa dapat diandalkan atau tidak jaringan yang mendasari dan layanan *network layer*. Karena itu, ISO telah mengembangkan suatu rumpun lima standar protokol *transport* yang masing-masing berorientasi terhadap layanan yang berbeda yang mendasari. Pada protokol TCP/IP, terdapat dua protokol *transport layer* yang umum, yaitu : TCP yang bersifat *connection-oriented* dan UDP yang bersifat *connectionless*.
- ***Network layer :*** menyediakan transfer informasi diantara ujung-ujung sistem melewati beberapa jairngan komunikasi berurutan. Ini membantu mengurangi beban pada *layer* tertinggi dari kebutuhan untuk mengetahui apapun mengenai transmisi data yang mendasari dan mengganti teknologi yang dipergunakan untuk menghubungkan sistem. Pada *layer* ini, sistem komputer berdialog dengan jaringan untuk menentukan alamat tujuan dan meminta fasilitas jaringan tertentu, misalnya prioritas. Namun ada kemungkinan untuk menghalangi fasilitas-fasilitas komunikasi yang dikelola oleh *network layer*,

misalnya, adanya *link* langsung dari titik ke titik di antara ujung sistem. Dalam hal ini, tidak diperlukan lagi *network layer* karena *data-link layer* mampu menyediakan fungsi yang diperlukan dalam mengelola *link*.

- ***Data-link layer :*** mengupayakan agar *link* fisik cukup baik dan menyediakan alat-alat untuk mengaktifkan, mempertahankan, dan menonaktifkan *link*. Layanan utama yang disediakan oleh *data-link layer* untuk *layer* yang lebih tinggi adalah *error detection* dan kontrol. Jadi, jika *layer* ini berfungsi dengan baik, maka *layer* di atasnya bisa menerima transmisi bebas kesalahan melewati *link*.
- ***Physical layer :*** mencakup *physical interface* di antara *device-device* dan aturan *bit-bit* dilewatkan dari satu ke yang lain. *Physical layer* memiliki empat karakteristik penting, yaitu :

  Mekanis : berkaitan dengan properti fisik dari *interface* ke media transmisi. Biasanya, spesifikasinya adalah dari konektor *pluggable* yang menggabungkan satu atau lebih *signal conductor*, yang disebut sirkuit.

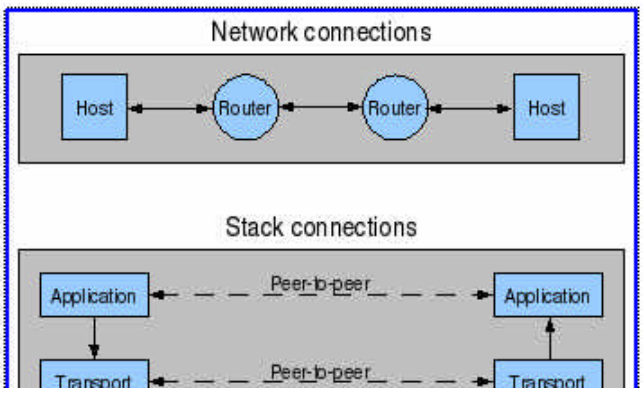
  Elektris : berkaitan dengan tampilan *bit-bit*.

  Fungsional : menentukan fungsi-fungsi yang ditampilkan oleh sirkuit tunggal dari *physical interface* diantara sebuah sistem dengan media transmisi.

  Prosedural : menentukan rangkaian kejadian dimana arus *bit* berpindah melalui medium fisik.

2.4.5. TCP/IP

Protokol TCP/IP merupakan hasil pengembangan dan riset protokol yang dilakukan atas eksperimental jaringan *paket-switched*, ARPANET, dan mendapat dukungan dana dari DARPA (*Defense Advanced Research Project Agency*). Secara umum, TCP/IP ini lebih dikenal sebagai TCP/IP *protocol suite*.



Gambar 2.17. Model Protokol TCP/IP

Sumber: Wikipedia, the free encyclopedia. *Internet protocol suite*. 2006. figure 1. <http://en.wikipedia.org/wiki/Internet_protocol_suite>

Model protokol TCP/IP terdiri dari empat *layer*, yaitu :

- ***Application layer :*** berisi logika yang dibutuhkan untuk mendukung aplikasi. Untuk jenis aplikasi yang berbeda diperlukan modul-modul terpisah yang sesuai dengan aplikasi tersebut.
- ***Transport layer :*** berfungsi memastikan seluruh data yang sampai di tujuan sesuai dengan data yang dikirim oleh pengirim. Oleh karena itu, *layer* ini harus memiliki mekanisme yang handal supaya data yang sampai di tujuan tidak ada yang hilang atau rusak. Mekanisme tersebut mencakup kegiatan seperti *flow control*, dan *error corection*.
- ***Network layer :*** berfungsi mengatur mekanisme pertukaran data antara dua ujung sistem yang dihubungkan ke jaringan yang berbeda. IP (*internet Protocol*) dipergunakan pada *layer* ini untuk menyediakan fungsi *routing* melintasi jaringan yang bermacam-macam. Protokol ini diterapkan tidak hanya pada ujung-ujung sistem, namun juga pada jalur-jalurnya.
- ***Data link layer :*** meliputi antarmuka fisik antara suatu perangkat transmisi data (komputer, *workstation*) dengan sebuah media transmisi atau jaringan. *Layer* ini berkaitan dengan karakteristik khusus dari media transmisi, sifat sinyal, *data rate*, dan hal-hal lain yang berkaitan dengan hal tersebut.

2.4.6. Hubungan antara *Winsock* dan Model OSI

Dalam organisasi internasional untuk standarisasi model *Open Systems Interconnection* (ISO/OSI), *winsock* beroperasi pada antarmuka *session layer* sampai *transport layer*. *Winsock* merupakan antarmuka antara aplikasi dengan *transport protocol* dan bekerja sebagai aliran data I/O.

Winsock menyederhanakan pengembangan aplikasi dalam *upper* ISO/OSI *layer* dengan menangani pertukaran data jaringan secara detail pada *lower layer*. *Winsock* menyediakan antarmuka yang dapat diprogram antara *upper layer* dengan *lower layer*. Aplikasi *winsock* sendiri berada pada *upper layer* (*application layer*, *presentation layer*, dan *session layer*). Pada aplikasi *winsock*, data dipaketkan dan dikirim melalui jaringan pada *lower layer* (*transport layer*, *network layer*, *data-link layer*, dan *physical layer*).

2.4.7. Fungsi-fungsi dalam *Winsock*

Setelah mengetahui tentang *winsock* dan arsitekturnya, sekarang akan dijelaskan fungsi-fungsi dalam *winsock* yang tentunya sangat perlu untuk diketahui agar bisa membangun sebuah aplikasi *client* / *server*.

2.4.7.1. Fungsi `accept`

Fungsi `accept` ini mempunyai tugas menerima koneksi yang berusaha masuk ke *socket*. Sintaknya sebagai berikut :

```
SOCKET accept (
   SOCKET s,
   struct sockaddr FAR *addr,
   int FAR *addrlen
   );
```

Parameter :

- `s` [in] menunjuk pada *socket* yang sedang mendengarkan koneksi setelah pemanggilan fungsi `listen`.
- `addr` [out] *optional pointer* ke *buffer* yang menerima alamat dari koneksi yang masuk, sebagai *communications layer*. Format `addr` ditentukan dengan *address family* yang dibuat ketika *socket* dari struktur SOCKADDR terbuat.
- `addrlen` [out] *optional pointer* ke integer yang berisi panjang dari `addr`.

Jika tidak terjadi kesalahan, maka `accept` akan mengembalikan nilai yang bertipe SOCKET yang merupakan deskripsi untuk *socket* yang baru. Jika ada kesalahan, maka fungsi ini akan mengembalikan nilai INVALID_SOCKET dan keterangan tentang kesalahan yang terjadi dapat diperoleh dengan memanggil fungsi `WSAGetLastError`.

Fungsi `accept` memproses koneksi pertama dalam antrian pada *socket* `s`. Fungsi `accept` kemudian menciptakan sebuah *socket* baru dan mengembalikan sebuah *handle* untuk *socket* tersebut. *Socket* baru inilah yang kemudian akan menangani koneksi yang sesungguhnya, dan *socket* ini memiliki *property* yang sama dengan *socket* `s`. Fungsi `accept` hanya digunakan pada koneksi jenis *connection-oriented* dengan *socket* `s` bertipe SOCK_STREAM.

2.4.7.2. Fungsi `bind`

Fungsi `bind` mempunyai tugas mengikatkan *socket* yang telah dibuat pada sebuah *local address* tertentu. Sintaknya sebagai berikut :

```
int bind (
   SOCKET s,
   const struct sockaddr FAR *name,
   int namelen
   );
```

Parameter :

- `s` [in] menunjuk pada *socket* yang belum diikatkan pada *local address*.
- `name` [in] alamat dari struktur SOCKADDR dimana *socket* akan ditempatkan.
- `namelen` [in] panjang nilai dari parameter `name`.

Jika tidak terjadi kesalahan, maka fungsi `bind` akan mengembalikan nilai nol. Sebaliknya jika terjadi kesalahan, maka fungsi ini akan mengembalikan nilai SOCKET_ERROR. Keterangan tentang kesalahan yang terjadi dapat diperoleh dengan memanggil fungsi `WSAGetLastError`.

Fungsi `bind` digunakan pada *socket* yang belum terkoneksi sebelum melakukan pemanggilan `connect` dan fungsi `listen`. Fungsi ini dipakai untuk mengikat semua jenis koneksi *socket*, baik koneksi *connection-oriented* ataupun *connectionless*.

2.4.7.3. Fungsi `closesocket`

Fungsi `closesocket` berfungsi untuk menutup keberadaan suatu *socket*. Sintaknya sebagai berikut :

```
int closesocket (
   SOCKET s
   );
```

Parameter :

- `s` [in] menunjuk pada *socket* yang akan ditutup.

Jika tidak ada kesalahan, maka fungsi `closesocket` akan mengembalikan nilai nol. Tetapi jika terjadi kesalahan, maka fungsi ini akan mengembalikan nilai SOCKET_ERROR dan keterangan tentang kesalahan yang terjadi dapat dilihat dengan memanggil fungsi `WSAGetLastError`.

2.4.7.4. Fungsi `connect`

Fungsi `connect` berfungsi membuat koneksi dengan spesifikasi tertentu. Sintaknya sebagai berikut :

```
int connect (
   SOCKET s,
   const struct sockaddr FAR *name,
   int namelen
   );
```

Parameter :

- `s` [in] menunjuk pada *socket* yang belum melakukan koneksi.
- `name` [in] nama *socket* dalam struktur SOCKADDR yang mana merupakan tempat koneksi akan dibuat.
- `namelen` [in] panjang nilai dari *parameter* `name`.

Jika tidak terjadi kesalahan, maka fungsi `connect` akan mengembalikan nilai nol. Sebaliknya jila terjadi kesalahan, maka fungsi ini akan mengembalikan nilai SOCKET_ERROR. Keterangan tentang kesalahan yang terjadi dapat diperoleh dengan memanggil fungsi `WSAGetLastError`.

Fungsi `connect` digunakan untuk menciptakan koneksi ke sebuah tujuan tertentu. Ketika koneksi sudah terbentuk, maka *socket* `s` siap untuk mengirim dan menerima data melalui koneksi yang terbentuk.

2.4.7.5. Fungsi `listen`

Fungsi `listen` berfungsi menempatkan *socket* pada suatu keadaan supaya bisa mendengarkan koneksi-koneksi yang datang. Sintaknya sebagai berikut :

```
int listen (
   SOCKET s,
   int backlog,
   );
```

Parameter :

- `s` [in] menunjuk pada *unbound socket*.
- `backlog` [in] menyatakan panjang maksimum antrian koneksi.

Jika tidak terjadi kesalahan, maka fungsi `listen` akan menghasilkan nilai nol. Sebaliknya jika terjadi kesalahan, maka fungsi ini akan mengembalikan nilai SOCKET_ERROR dan untuk mendapatkan keterangan tentang kesalahannya dapat menggunakan fungsi `WSAGetLastError`.

Untuk menerima koneksi, sebuah *socket* harus terlebih dahulu diciptakan dengan menggunakan fungsi `socket` dan diikat pada suatu alamat dengan menggunakan fungsi `bind`. Kemudian fungsi `listen` bertugas mendengar koneksi yang datang pada *socket* dan fungsi `accept` yang akan menerima koneksi yang datang tersebut. Jenis koneksi yang menggunakan fungsi `listen` untuk mendengarkan koneksi dari *client* biasanya merupakan *socket* yang diciptakan oleh fungsi `socket` dengan tipe SOCK_STREAM dan bersifat *connection-oriented*.

2.4.7.6. Fungsi `recv`

Fungsi `recv` berfungsi untuk menerima data dari *socket* yang terhubung. Sintaknya sebagai berikut :

```
int recv (
   SOCKET s,
   char FAR *buf,
   int len,
   int flags
   );
```

Parameter :

- `s` [in] menunjuk pada *socket* yang terkoneksi.
- `buf` [out] *buffer* untuk data yang diterima.
- `len` [in] panjang dari *buffer*.
- `flags` [in] merupakan cara pemanggilan dibuat.

Jika tidak terjadi kesalahan, maka fungsi ini akan mengembalikan jumlah *byte* yang diterima dan akan mengembalikan nilai SOCKET_ERROR jika terjadi kesalahan. Keterangan tentang kesalahan yang terjadi dapat diperoleh dengan pemanggilan fungsi `WSAGetLastError`. Jika koneksi terputus, maka fungsi ini akan mengembalikan nilai nol.

Fungsi `recv` digunakan untuk membaca data yang masuk pada *socket*, baik yang terkoneksi dengan protokol *connection-oriented* ataupun *connectionless*. Ketika menggunakan protokol *connection-oriented*, sebuah *socket*

harus memiliki koneksi dulu sebelum memanggil fungsi `recv`. Sedangkan jika menggunakan protokol *connectionless*, sebuah *socket* harus diikat ke suatu alamat tertentu sebelum memanggil fungsi `recv`.

Untuk *socket connection-oriented* pemanggilan fungsi `recv` akan mengembalikan sejumlah informasi yang tersedia sebatas ukuran *buffer* yang telah ditentukan. Jika sisi *remote* telah menghentikan koneksi, dan semua data telah diterima, fungsi `recv` dengan segera akan melengkapinya dengan 0 *byte*.

2.4.7.7. Fungsi `send`

Fungsi `send` berfungsi untuk mengirimkan data ke *socket* yang terhubung. Sintaknya sebagai berikut :

```
int send (
   SOCKET s,
   const char FAR *buf,
   int len,
   int flags,
   );
```

Parameter :

- `s` [in] menunjuk pada *socket* yang terkoneksi.
- `buf` [out] *buffer* berisi data yang akan dikirimkan.
- `len` [in] panjang dari *buffer*.
- `flags` [in] merupakan cara pemanggilan dibuat.

Jika tidak terjadi kesalahan pada waktu pemanggilan fungsi ini, maka fungsi ini akan mengembalikan jumlah *byte* yang telah dikirimkan, yang mana besarnya dapat kurang dari angka yang disebutkan pada parameter **`len`** untuk model *socket nonblocking*. Sebaliknya, jika terjadi kesalahan maka fungsi ini akan mengembalikan nilai SOCKET_ERROR, dan kode kesalahan yang terjadi dapat diketahui dengan memanggil fungsi `WSAGetLastError`.

Pemanggilan fungsi `send` dengan parameter **`len`** berisi nilai nol diperbolehkan. Dalam banyak kasus, fungsi `send` akan mengembalikan nilai nol sebagai nilai yang sah.

2.4.7.8. Fungsi `shutdown`

Fungsi `shutdown` berfungsi untuk mematikan kemampuan sebuah *socket* untuk mengirim dan menerima data. Sintaknya sebagai berikut :

```
int shutdown (
   SOCKET s,
   int how
   );
```

Parameter :

- `s` [in] menunjuk pada sebuah *socket.*
- `how` [in] menunjuk pada tipe operasi apa saja yang tidak lagi dapat dijalankan.

Jika tidak terjadi kesalahan, maka fungsi `shutdown` akan mengembalikan nilai nol. Jika terjadi kesalahan, maka fungsi ini akan mengembalikan nilai SOCKET_ERROR, dan keterangan tentang kesalahan yang terjadi dapat diketahui dengan pemanggilan fungsi `WSAGetLastError`.

Fungsi `shutdown` digunakan oleh semua tipe *socket* untuk mematikan kemampuan menerima dan mengirimkan data. Jika parameter `how` adalah SD_RECEIVE, akan mengakibatkan pemanggilan fungsi `recv` pada socket tidak lagi diijinkan. Hal ini tidak mempengaruhi *layer-layer* protokol di bawahnya.

Jika parameter `how` adalah SD_SEND, akan mengakibatkan pemanggilan fungsi `send` tidak lagi diperbolehkan. Untuk *socket* TCP, FIN akan dikirimkan setelah semua data dikirim dan menerima pemberitahuan dari penerima. Sedangkan pemberian nilai SD_BOTH ke parameter `how` akan melumpuhkan fungsi `recv` dan fungsi `send` secara bersamaan.

Pemanggilan fungsi ini tidak akan menutup *socket*. Penutupan *socket* akan dilakukan jika terjadi pemanggilan fungsi `closesocket`. Untuk lebih meyakinkan bahwa semua data telah dikirim dan diterima pada *socket* yang terhubung sebelum *socket* tersebut ditutup, sebuah aplikasi haruslah memanggil fungsi `shutdown` untuk menutup koneksi sebelum benar-benar menutup *socket* dengan memanggil fungsi `closesocket`.

Meskipun pemanggilan fungsi `shutdown` tidak menutup *socket* yang terhubung, namun tidak dianjurkan untuk menggunakan kembali *socket* tersebut. Pada umumnya, winsock tidak mendukung pemakaian fungsi `connect` pada *socket* yang telah melakukan pemanggilan fungsi `shutdown`.

2.4.7.9. Fungsi `socket`

Fungsi `socket` digunakan untuk membuat sebuah *socket* yang akan diikatkan pada *service provider* tertentu. Sintaknya sebagai berikut :

```
SOCKET socket (
   int af,
   int type,
   int protocol
   );
```

Parameter :

- `af` [in] alamat *family* yang telah ditentukan.
- `type` [in] tipe untuk *socket* baru. Tipe *socket* ada 2 macam, SOCK_STREAM dan SOCK_DGRAM. Tipe SOCK_STREAM umumnya digunakan untuk koneksi TCP, dan tipe SOCK_DGRAM untuk koneksi UDP.
- `protocol` [in] digunakan dengan *socket* yang secara khusus untuk menunjukkan alamat *family*.

Jika tidak terjadi kesalahan, maka fungsi `socket` akan mengembalikan sebuah referensi yang menunjuk ke *socket* baru yang telah dibuat. Jika fungsi `socket` gagal membuat sebuah *socket* baru, maka fungsi ini akan mengembalikan nilai INVALID_SOCKET. Keterangan tentang kesalahan yang terjadi pada fungsi ini akan diperoleh dengan memanggil fungsi `WSAGetLastError`.

Tipe *socket* SOCK_STREAM menyediakan koneksi *full-duplex*, dan tiap *socket* harus benar-benar terhubung sebelum proses pertukaran data dilakukan. Koneksi ke *socket* yang lain dibuat dengan memanggil fungsi `connect`. Setelah terhubung, data dapat ditransfer dengan menggunakan fungsi `send` dan fungsi

`recv`. Untuk mengakhiri koneksi dapat dilakukan dengan memanggil fungsi `closesocket`.

2.4.8. Manipulasi Jendela Program di Windows

Windows menyediakan berbagai macam fungsi yang dapat digunakan untuk keperluan membuat jendela aplikasi, dan berbagai macam manipulasinya. Biasanya pembuatan jendela aplikasi di Windows melibatkan pembuatan *class* dan *title* dari jendela aplikasi tersebut. Saat sebuah jendela aplikasi dibuat, Windows juga akan memberikan ID yang disebut dengan PID (*Process* ID) untuk proses yang melibatkan aplikasi tersebut.

2.4.8.1. Fungsi `EnumWindows`

Fungsi ini digunakan untuk mengenumerasi semua jendela *top-level* yang aktif pada layar dengan cara memberikan *handle* jendela aplikasi yang ada ke fungsi *callback* yang dibuat oleh pengguna. Fungsi ini akan terus berjalan sampai semua jendela *top-level* yang ada dienumerasi. Sintaknya sebagai berikut :

```
BOOL EnumWindows (
   WNDENUMPROC lpEnumFunc,
   LPARAM lparam
   );
```

Parameter :

- `lpEnumFunc` [in] adalah *pointer* ke fungsi *callback* yang dibuat oleh pengguna.
- `lparam` [in] menspesifikasikan nilai yang akan diberikan ke dalam fungsi *callback*.

Jika fungsi ini sukses dilakukan, maka nilai balik yang diberikan bernilai selain nol, sebaliknya jika *error* maka nilai baliknya adalah nol. Jika parameter `lpEnumFunc` terjadi *error*, maka fungsi `EnumWindows` ini juga akan menghasilkan nilai nol. Untuk dapat mengetahui *error* yang terjadi, maka pada parameter `lpEnumFunc` harus memanggil fungsi `SetLastError`.

2.4.8.2. Fungsi `EnumWindowsProc`

Fungsi ini adalah fungsi *callback* yang didefinisikan oleh user untuk digunakan dalam fungsi `EnumWindows` atau `EnumDesktopWindows`. Fungsi ini digunakan untuk menerima *handle* jendela *top-level* yang sedang aktif. Sintaknya sebagai berikut :

```
BOOL CALLBACK EnumWindowsProc (
   HWND hwnd,
   LPARAM lparam
   );
```

Parameter :

- `hwnd` [in] merupakan *handle* dari jendela *top-level*
- `lparam` [in] menspesifikasikan nilai yang diberikan oleh fungsi `EnumWindows` atau `EnumDesktopWindows`.

Selama nilai *return*-nya bernilai TRUE, fungsi ini akan terus mengenumerasi jendela *top-level* yang ada. Agar fungsi berhenti, nilai *return*-nya harus diset FALSE.

2.4.8.3. Fungsi `FindWindow`

Fungsi ini digunakan untuk menampung nilai *handle* dari suatu jendela. Sintaknya sebagai berikut :

```
HWND FindWindow (
   LPCTSTR lpClassName,
   LPCTSTR lpWindowName
   );
```

Parameter :

- `lpClassName` [in] merupakan *pointer* ke *string* yang menyatakan *class* jendela aplikasi. Apabila parameter ini diset NULL, maka fungsi akan mencari jendela dengan menggunakan parameter `lpWindowName`.

- `lpWindowName` [in] merupakan parameter yang menspesifikasikan nama jendela aplikasi yang dicari. Apabila parameter ini NULL, maka semua window yang aktif akan diterima *handle*-nya.

Jika fungsi ini sukses dilakukan, maka nilai balik yang didapatkan adalah *handle* jendela aplikasi yang sesuai dengan parameter `lpClassName` dan `lpWindowName`. Jika fungsi *error*, maka nilai baliknya adalah NULL. Untuk mengetahui *error* yang terjadi kita bisa menggunakan fungsi `GetLastError`.

2.4.8.4. Fungsi `GetWindowText`

Fungsi ini digunakan untuk mendapatkan teks judul jendela aplikasi. Sintaknya sebagai berikut :

```
int GetWindowText (
   HWND hwnd,
   LPTSTR lpString,
   int nMaxCount
   );
```

Parameter :

- `hwnd` [in] merupakan *handle* dari jendela aplikasi yang akan diterima judulnya.
- `lpString` [out] merupakan *pointer* ke *buffer* yang akan menerima teks judul jendela aplikasi.
- `nMaxCount` [in] menspesifikasikan jumlah maksimum karakter yang akan dikopi ke *buffer*.

2.4.8.5. Fungsi `SetForegroundWindow`

Fungsi ini digunakan untuk mengaktifkan jendela aplikasi yang diinginkan. Sintaknya sebagai berikut :

```
BOOL SetForegroundWindow (
   HWND hwnd
   );
```

Parameter :

- `hwnd` [in] merupakan *handle* jendela aplikasi yang ingin diaktifkan. Jika fungsi berhasil, akan memberikan nilai balik selain nol.

2.4.9. Manipulasi *Input Keyboard* dan *Mouse*

Windows juga mempunyai fungsi yang dapat digunakan untuk keperluan manipulasi *input* dari pengguna. Manipulasi *input* tersebut adalah manipulasi *input keyboard* dan *mouse*. Dengan menggunakan fungsi ini kita dapat mengirimkan perintah *keyboard* atau *mouse* ke jendela aplikasi yang aktif.

2.4.9.1. Fungsi `keybd_event`

Fungsi ini digunakan untuk mengemulasikan kerja dari *keyboard*. Sintaknya sebagai berikut :

```
VOID keybd_event (
   BYTE bVk,
   BYTE bscan,
   DWORD dwFlags,
   PTR dwExtraInfo
   );
```

Parameter :

- `bVk` [in] menspesifikasikan kode *virtual-key* dari *keyboard*. Kode ini bernilai 1 sampai 254
- `bScan` parameter ini tidak digunakan.
- `dwFlags` [in] menspesifikasikan bermacam aspek yang dilakukan dalam pengeksekusian fungsi. Parameter ini dapat bernilai sebagai berikut :

  KEYEVENTF_EXTENDEDKEY : Jika nilai ini dispesifikasikan, kode *scan* yang didapatkan mempunyai *prefix* BYTE bernilai 0xE0 (224).

  KEYEVENTF_KEYUP : Jika nilai ini dispesifikasikan, maka emulasi *keyboard* yang dilakukan adalah pelepasan tombol. Jika tidak dispesifikasikan, maka emulasi yang dilakukan adalah penekanan tombol.

2.4.9.2. Fungsi `mouse_event`

Fungsi ini digunakan untuk mengemulasikan kerja dari *mouse*, baik itu penekanan tombol *mouse* maupun gerakan *pointer mouse.* Sintaknya sebagai berikut :

```
VOID mouse_event (
   DWORD dwFlags,
   DWORD dx,
   DWORD dy,
   DWORD dwData,
   ULONG_PTR dwExtraInfo
   );
```

Parameter :

- `dwFlags` [in] menspesifikasikan berbagai macam aspek pengemulasian kerja *mouse*. Parameter ini dapat bernilai kombinasi dari nilai berikut :

  MOUSEEVENTF_ABSOLUTE : Menspesifikasikan bahwa parameter `dx` dan `dy` merupakan koordinat absolut. Jika parameter ini tidak diberikan, maka nilai `dx` dan `dy` merupakan nilai pergerakan *pointer mouse* dihitung dari posisi *pointer mouse* yang terakhir. Jika nilai `dx` negatif berarti *pointer mouse* bergerak ke kiri. Jika nilai `dy` negatif berarti *pointer mouse* bergerak ke atas, dan sebaliknya.

  MOUSEEVENTF_MOVE : Menspesifikasikan bahwa *mouse* akan digerakkan.

  MOUSEEVENTF_LEFTDOWN : Menspesifikasikan bahwa tombol kiri *mouse* ditekan.

  MOUSEEVENTF_LEFTUP : Menspesifikasikan bahwa tombol kiri *mouse* dilepas.

  MOUSEEVENTF_RIGHTDOWN : Menspesifikasikan bahwa tombol kanan *mouse* ditekan.

  MOUSEEVENTF_RIGHTUP : Menspesifikasikan bahwa tombol kanan *mouse* ditekan.

  MOUSEEVENTF_MIDDLEDOWN : Menspesifikasikan bahwa tombol tengah *mouse* ditekan.

  MOUSEEVENTF_MIDDLEUP : Menspesifikasikan tombol tengah *mouse* dilepas.

MOUSEEVENTF_WHEEL : Menspesifikasikan bahwa tombol *wheel mouse* digerakkan. Jumlah gerakan yang dilakukan dispesifikasikan di parameter `dwData`.

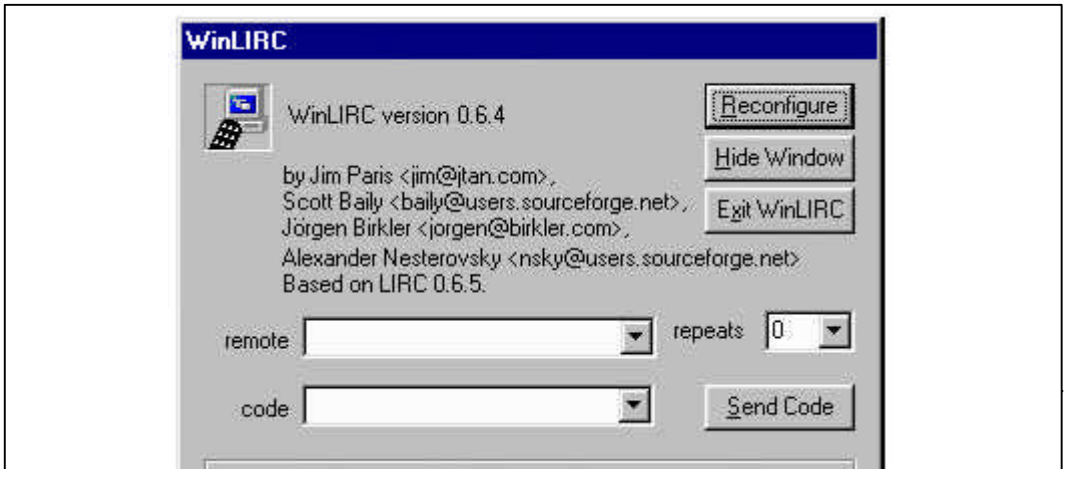
MOUSEEVENTF_XDOWN : Menspesifikasikan tombol X *mouse* ditekan.

MOUSEEVENTF_XUP : Menspesifikasikan tombol X *mouse* dilepas.

- `dx` [in] menspesifikasikan posisi absolut horisontal *pointer mouse* jika parameter MOUSEEVENTF_ABSOLUTE diset atau gerakan horisontal *mouse* jika parameter MOUSEEVENTF_ABSOLUTE tidak diset.
- `dy` [in] menspesifikasikan posisi absolut vertikal *pointer mouse* jika parameter MOUSEEVENTF_ABSOLUTE diset atau gerakan vertikal *mouse* jika parameter MOUSEEVENTF_ABSOLUTE tidak diset.
- `dwData` [in] jika parameter MOUSEEVENTF_WHEEL diset, maka parameter ini menspesifikasikan jumlah gerakannya. Jika parameter MOUSEEVENTF_XDOWN atau MOUSEEVENTF_XUP diset, maka parameter ini menentukan tombol X mana yang ditekan.
- `dwExtraInfo` [in] menspesifikasikan nilai tambahan yang digunakan untuk melakukan emulasi *mouse.*

2.4.10. WinLIRC *Server*

Software opensource ini dapat kita gunakan untuk mengirim dan menerima sinyal *infrared remote* kontrol standar melalui komputer. *Software* ini merupakan versi *porting* dari *software opensource* di Linux yang bernama LIRC (*Linux Infrared Remote Control*). Tidak seperti LIRC yang kompatibel dengan beberapa macam *interface infrared*, WinLIRC saat ini hanya dapat memakai *interface* serial untuk dapat menggunakan komunikasi *infrared remote* kontrol. Tampilan utama WinLIRC dapat dilihat pada gambar 2.17.



Gambar 2.18. Tampilan Utama WinLIRC

Sumber: Paris, Jim. *WinLIRC Quick Overview.* 9 March 2006. figure 1 <http://winlirc.sourceforge.net/overview.html>

Masing-masing kegunaan dari menu tampilan utama tersebut adalah sebagai berikut :

- Tombol *Reconfigure*

  Digunakan untuk menampilkan dialog konfigurasi program.
- Tombol *Hide Window*

  Digunakan untuk menghilangkan dialog program dari *desktop*.
- *Combobox remote*

  Digunakan untuk memilih konfigurasi *remote* yang digunakan untuk mengirimkan sinyal *infared*.
- *Combobox code*

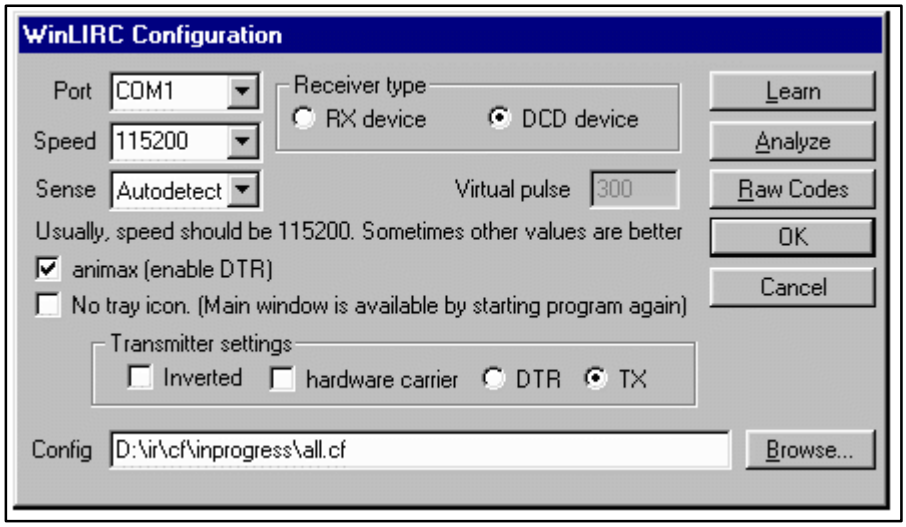
  Digunakan untuk memilih kode *remote* yang ingin dikirim.
- *Combobox repeats*

  Digunakan untuk memilih berapa banyak perulangan penekanan tombol yang ingin dilakukan.

- Tombol *send*

  Digunakan untuk mengirim sinyal *infrared*.

Untuk dapat mulai menggunakan *software* ini, perlu dilakukan konfigurasi terlebih dulu, seperti pemilihan *port* serial yang ingin dipakai, dan konfigurasi *remote* kontrol yang ingin dikirim atau diterima kodenya. Tampilan jendela konfigurasi WinLIRC dapat dilihat pada gambar 2.18.



Gambar 2.19. Jendela Konfigurasi WinLIRC

Sumber: Paris, Jim. *WinLIRC Quick Overview.* 9 March 2006. figure 2 <http://winlirc.sourceforge.net/overview.html>

Fungsi dari masing-masing menu konfigurasi di atas dapat dilihat kegunaannya sebagai berikut :

- *Combobox Port*

  Digunakan untuk mengatur serial *port* yang akan digunakan.
- *Combobox Speed*

  Digunakan untuk mengatur kecepatan transmisi sinyal yang dikirim atau diterima.
- *Combobox sense*

  Digunakan untuk mengatur apakah *receiver* aktif *high* atau *low.*

- *Receiver type*

  Digunakan untuk memilih apakah *receiver* menggunakan pin RX atau DCD.
- *Editbox Virtual pulse*

  Digunakan untuk mengatur panjang pulsa untuk *receiver* RX.
- *Checkbox animax*

  Digunakan untuk men-*set* pin DTR menjadi aktif *high*.
- *Checkbox No tray icon*

  Digunakan untuk menyembunyikan *icon tray*.

- *Transmitter setting*

  Digunakan untuk mengatur tipe *transmitter* yang dipakai.
- *Checkbox inverted*

  Digunakan untuk membalik logika data sinyal yang dikirim.
- *Checkbox hardware carrier*

  Digunakan untuk mencegah WinLIRC membuat frekuensi *carrier*.
- *Radiobox* DTR

  Untuk menspesifikasikan *transmitter* menggunakan pin DTR. Penggunaan *transmitter* ini tidak disarankan.
- *Radiobox* TX

  Untuk menspesifikasikan bahwa *transmitter* menggunakan pin TX.
- *Editbox Config*

  Untuk menspesifikasikan *full path* dari file konfigurasi.
- *Browse*

  Digunakan untuk mencari *file* konfigurasi yang akan digunakan.
- *Learn*

  Digunakan untuk mendapatkan *input* sinyal dari *remote* kontrol untuk dipelajari.
- *Analyze*

  Digunakan untuk mempelajari sinyal yang telah diterima pada proses *learn*.
- *Raw Codes*

  Digunakan untuk menampilkan *timing raw* data yang diterima dari *remote*.
- *OK*

  Untuk menyimpan *file* konfigurasi ke *registry.*
- *Cancel*

  Untuk membatalkan konfigurasi yang telah dilakukan.

2.4.10.1. Format Data yang dikirimkan WinLIRC

Untuk dapat memanfaatkan program *server* WinLIRC ini harus digunakan aplikasi lain yang terhubung dengan program ini. Aplikasi tersebut dapat berupa *plugin* untuk program yang sudah ada ataupun aplikasi independen

yang ditulis untuk mengolah sinyal *infrared* yang diterima atau dikirim oleh WinLIRC.

Server WinLIRC ini beroperasi pada *port* TCP/IP nomor 8765. WinLIRC diklaim dapat menerima sampai 16 *client* pada saat yang bersamaan. Saat sebuah sinyal *infrared* berhasil dikodekan, *server* ini akan mengirimkan sebaris teks ASCII ke semua *client* yang terhubung dengannya. Contoh format sinyal yang dikirim *server* ini adalah sebagai berikut :

```
0000000000eab154 00 play myremote
0000000000eab154 01 play myremote
0000000000eab154 02 play myremote
0000000000eab154 03 play myremote
```

Gambar 2.20. Format Data WinLIRC

Sumber: Paris, Jim. *WinLIRC Developer Information*. 9 March 2006. p.1. <http://winlirc.sourceforge.net/developer.html>

Data ini mempunyai empat bagian. Bagian pertama merupakan kode heksadesimal tombol *remote* yang diterima. Bagian kedua merupakan perulangan tombol yang ditekan. Bagian ketiga merupakan nama tombol yang ditekan. Dan bagian terakhir merupakan nama *remote* kontrol yang diterima.