รายการอ้างอิง

ภาษาไทย

ยืน ภู่วรวรรณ และ ไพศาล สงวนหมู่. 2521. การสื่อสารข้อมูลและไมโครคอมพิวเตอร์เน็ตเวอร์ค. กรุงเทพมหานคร:สำนักพิมพ์ซีเอ็ดยูเคชัน.

ภาษาอังกฤษ

Miller, M.A. 1992. TroubleShooting TCP/IP. USA: Prentice Hall.

Moore, M..L. 1993. The Ultimate Sound Blaster Book .USA: Que Corporation.

Nance, B. 1990. Network Programaing in C. USA : Que Corporation.

Schwaderer, W.D. 1992. C Programmer's Guide to NetBIOS, IPX and SPX. Indiana:Sams Publishing.

Stallings, W. 1991. Data and Computer Communication. USA:Macmillan Publishing.

ภาคผนวก

ภาคผนวก ก

การเขียนโปรแกรมเกี่ยวกับเครือข่ายเฉพาะที่

ในการเขียนโปรแกรมเกี่ยวกับเครือข่ายเฉพาะที่ มีเครื่องมือให้ใช้อยู่หลายชนิด ในที่นี้จะใช้ IPX เป็นเครื่องมือในการพัฒนาโปรแกรม เพราะเป็นเครื่องมือในการพัฒนาโปรแกรม สำหรับระบบปฏิบัติการเครือข่าย NetWare และมีฟังก์ชั่นให้เลือกใช้ได้มาก ตลอดจนมีตำราสนับสนุนมาก

IPX เป็นบริการแบบคาด้าแกรม ซึ่งมีให้ใช้บนระบบข่ายงานเฉพาะที่ที่ใช้ Novell NetWare เป็นระบบปฏิบัติการของเครือข่าย ตารางต่อไปนี้ จะทำการเปรียบเทียบ โอเปอเรชั่นเกี่ยวกับไฟล์ของดอส กับ โอเปอเรชั่น ที่เทียบเท่ากับของ IPX

โอเปอเรชั่นของแฟ้ม	โอเปอเรชั่นใน IPX
Open	Open Socket
Read	Listen for Packet
Write	Send Packet
Seek	--
Close	Close Socket

ตาราง ก.1 ปฏิบัติการของ IPXเทียบกับปฏิบัติการเรื่องแฟ้มของดอส

ทดสอบการติดตั้ง IPX

ในการทดสอบว่ามี IPX ถูกติดตั้งอยู่หรือไม่ จะต้องใช้อินเทอรัพต์ 2F ซึ่งทำได้โดยการใส่ค่า 0X7A ไปในรีจิสเตอร์ AH และใส่ค่า 0 ในรีจิสเตอร์ AL แล้วสั่งอินเตอร์รัพต์ 2F ถ้ามี IPX ติดตั้งอยู่แล้วจะได้รับค่า 0XFF คืนมาในรีจีสเตอร์ AL และจะได้จุดเรียก IPX และ SPX คืนมาในคู่รีจิสเตอร์ ES:DI โดยที่จะเป็นพอยเตอร์แบบ FAR ที่ชี้ไปยังฟังก์!ชั่น ฟังก์ชัน IPX_Installed() ต่อไปนี้เป็นการแสดงให้เห็นการทดสอบ IPX ฟังก์!ชันนี้จะคืน -1 มาให้ถ้าไม่มี IPX ติดตั้งอยู่ และจะคืนค่า 1 ถ้ามี IPX ติดตั้งอยู่ และฟังก์!ชันนี้จะเตรียมฟังก์!ชันพอยเตอร์ *IPX_SPX ให้ด้วย ซึ่งเราสามารถใช้ได้สำหรับเรียกใช้ IPX

การเรียกใช้ IPX

การเรียกใช้ IPX ทำได้ 2 วิธี และแต่ละวิธี ก็มีทั้งข้อดีและข้อเสีย สำหรับวิธีแรก จะต้องทำการใส่ค่าต่างๆ ให้แก่รีจิสเตอร์ ตามที่จำเป็นสำหรับแต่ละฟังก์ชั่น จากนั้นก็เรียก IPX_SPX ฟังก์ชั่นพอยเตอร์ซึ่งได้มาจากการติดตั้งโปรแกรม (Install) วิธีนี้จะทำได้ง่ายในเทอร์โบซี

ตัวอย่างการเรียกให้ IPX ใน TURBO C

```
Void ipx_Listen_for_packet(struct ECB *ecb_ptr)
{
  _ES = FP_SEG( (void far *) ecb_ptr);
  _SI = FP_OFF( (void far *) ecb_ptr);
  _BX = 0x0004;
  ipx_spx();
}
```

วิธีที่สองคือ การใช้อินเทอรัพต์ 7A ในการเรียก IPX อินเทอรัพต์เบอร์นี้ สามารถใช้เป็นจุดเรียก IPX ได้ วิธีนี้เหมาะกับผู้ที่ใช้ ไมโครซอฟท์ซี ตัวอย่างการเรียกใช้ฟังก์ชั่นของ IPX โดยใช้อินเทอรัพต์ 7A ในไมโครซอฟต์ซี

```
void close_socket(unsigned socket)
{
  union REGS regs;
  regs.x.bx = 0x0001;
  regs.x.dx = socket;
  int86(0x7A,&regs,&regs);
}
```

ข้อเสียของอินเทอรัพต์ 7A คือ มันถูกใช้ ในการทำอีมูเลเตอร์สำหรับ IBM 3270 และถูกใช้ ในซอฟต์แวร์อื่นๆ บางตัว เพราะฉะนั้นโปรแกรมที่เขียนนี้ เราจะรันได้ก็ต่อเมื่อเรามั่นใจได้ว่า จะไม่รันพร้อมกับโปรแกรมอื่นๆ ที่ใช้อินเทอรัพต์เบอร์นี้

สำหรับการทำวิทยานิพนธ์ฉบับนี้ จะใช้วิธีการแรก คือเรียกใช้ IPX_SPX ที่ได้มาจากการติดตั้งโปรแกรม

Event Control Block

Event Control Block หรือ ECB จะทำหน้าที่คล้ายกับ Network Control Block ในเรื่อง NetBIOS ECBจะไม่ถูกส่งไปในเครือข่าย แต่ECB จะทำหน้าที่เหมือนกับ กลุ่มของคำแนะนำให้กับ IPX และเป็นที่เก็บของให้ IPX ด้วย เราควรจะสร้าง ECB แยกกัน สำหรับแต่ละปฏิบัติการของ IPX ที่เราจะมีทั้งหมดในโปรแกรม ตัวอย่างต่อไปนี้แสดงให้เห็นโครงสร้างของ ECB

```
struct ECB
{
   void far          *link_address;
   void far          (*event_service_routine)(void);
   unsigned char  in_use;
   unsigned char   completion_code;
   unsigned int     socket_number;
   unsigned char   ipx_workspace[4];
   unsigned char   driver_workspace[12];
   unsigned char   immediate_address[6];
   unsigned int     packet_count;
   struct{
      void far        *address;
      unsigned  int  length;
      }packet[2];
};
```

ฟิลด์ Event_Eervice_Routine (ESR) เป็นพอยเตอร์ชี้ไปยังฟังก์ชั่น แฟล็กชื่อ in_use จะมีค่าไม่เป็นศูนย์ ถ้า IPX กำลังทำงานอะไรบางอย่าง อยู่ในเบื้องหลัง (Background) ซึ่งเราสามารถตรวจสอบว่า งานเสร็จหรือยัง โดยการทำลูปเพื่อโพล (Poll) จนกระทั่ง In_use กลายเป็น 0 และเมื่อเหตุการณ์ (Event) เสร็จสิ้นแล้ว Completion_code จะเก็บค่าที่ได้คืนมาจาก IPXสำหรับเหตุการนั้นๆ เราใช้ Socket_Number เพื่อระบุซอคเก็ต (Socket) ที่เราได้เปิดไว้และกำลังใช้ส่งข้อมูลอยู่ สำหรับการส่งจะต้องตั้งค่าฟิลด์ Immediate_Address ให้มีค่าเป็นเน็ตเวอร์คแอดเดรส ของบริดจ์ ที่ข้อมูลจะต้องวิ่งผ่าน สำหรับกลุ่มสุดท้ายมี Packet_Count Packet[ ].address และ Packet[ ].length จะบอก IPX เกี่ยวกับ Data Area ซึ่ง IPX

จะต้องนำมารวมกัน ก่อนส่งข้อมูลลงเครือข่าย หรือ ต้องแยกข้อมูลออก หลังจากได้รับข้อมูลจากเครือข่าย เพื่อส่งให้แอพพลิเคชั่น

เฮดเดอร์ของ IPX แพ็กเก็ต

30 ไบต์แรกของแต่ละแพ็กเก็ตที่เดินทางข้ามเครือข่ายจะใช้เป็น แพ็กเก็ตเฮดเดอร์ (Packet Header) IPX จะจัดการบางฟิลด์ด้วยตัวของมันเอง ฟิลด์ที่เราจะต้องลงไปยุ่งเกี่ยวก็คือ Packet_type (ต้องใส่ค่าเป็น 4) และฟิลด์ที่เกี่ยวกับปลายทาง คือ Dest_network_number Dest_network_node และ Dest_network_socket

```
struct IPXHEADER
{
  unsigned int    checksum;
  unsigned int    length;
  unsigned char   transport_control;
  unsigned char   packet_type;
  unsigned char   dest_network_number[4];
  unsigned char   dest_network_node[6];
  unsigned int    dest_network_socket;
  unsigned char   source_network_number[4];
  unsigned char   source_network_node[6];
  unsigned int    source_network_socket;
};
```

คำสั่งใน IPX

เมื่อเราต้องการใช้ IPX ให้ใส่เบอร์ของฟังก์ชั่นที่ต้องการใช้งาน ในรีจีสเตอร์ BX ถ้าฟังก์ชั่นใดจำเป็นต้องใช้ ECB ให้ใส่พอยเตอร์ที่ชี้ไปที่ ECB ลงในรีจิสเตอร์ ES:SI ต่อไปนี้เป็นฟังก์ชั่นของ IPX

ชื่อฟังก์ชัน	เบอร์ฟังก์ชัน
IPX_OpenSocket	0x00
IPX_Close_Socket	0x01
IPX_Get_Local_Target	0x02
IPX_Send_Packet	0x03
IPX_Listen_For_Packet	0x04
IPX_Schedule_IPX_Event	0x05
IPX_Cancel_Event	0x06
IPX_Get_Interval_Marker	0x08
IPX_Get_Internetwork_Address	0x09
IPX_Relinquish_Control	0x0A
IPX_Disconnect_From_Target	0x0B

ตาราง ก.2 ฟังก์ชันของ IPX

Event Service Routine

Event Service Routine (ESR) สำหรับ IPX จะทำหน้าที่เหมือนกับ Post Routine ใน NetBIOS เมื่อทำงานตามคำสั่งเสร็จ IPX จะเรียก ESR ที่เราบอกไว้ ถ้าเราใส่ Null Pointer ในตำแหน่งของ Event_Service_Routine ในECB IPX จะไม่เรียก ESR เมื่อทำงานเสร็จ ถ้าจะทำงานในลักษณะที่ไม่มี ESR เช่นนี้ จะต้องไม่ลืมที่จะให้โอกาสแก่ IPX เพื่อให้ทำงานให้เสร็จก่อนที่จะให้งานใหม่ ตัวอย่างต่อไปเป็นการวนตรวจสอบว่า IPX ทำงานเสร็จหรือยัง

```
while(receive_ecb.in_use)
  ipx_relinquish_control();
```

ในกรณีที่เราระบุ ESR ไว้ในโครงสร้าง ECB ฟังก์ชั่นจะถูกเรียกโดยที่เงื่อนไขต่อไปนี้เป็นจริง

-อินเทอรัพต์ถูกปิด (DISABLE)

-ฟังก์ชั่นจะถูกเรียกใช้แบบ Far Call (เรียกแบบข้ามเซ็กเมนต์ได้)

-พอยเตอร์ที่ชี้ไปยัง ECB จะอยู่ในคู่รีจิสเตอร์ ES:SI

-รีจิสเตอร์ต่างๆ จะถูกเก็บค่าไว้

-รีจิสเตอร์ DS ไม่จำเป็นจะต้องชี้ไปยังพื้นที่เก็บข้อมูลของโปรแกรม

-แฟล็กที่อยู่ในโครงสร้าง ECB ที่ชื่อ in_use จะถูกเซ็ตค่าเป็น 0

-การเรียกใช้ฟังก์ชั่นของดอส เช่น เปิดแฟ้มเพื่อเขียน หรืออ่านอาจเป็นการไม่ปลอดภัย

การใช้ ESR ในเทอร์โบซีนั้น จะสะดวกกว่า ที่จะใช้ไมโครซอฟต์ซี เพราะเทอร์โบซียอมให้เราเข้าถึงรีจิสเตอร์ของซีพียูได้โดยตรง เราสามารถเซ็ตค่าในรีจีสเตอร์ DS ให้ชี้ไปยังพื้นที่เก็บข้อมูล ของโปรแกรม และสร้าง Far Pointer ให้ชี้ไปยังโครงสร้าง ECB จากคู่รีจิสเตอร์ ES:SI ดังตัวอย่างต่อไปนี้

```
struct   ECB far *completed_ecb_ptr;
int      event_completed_flag;
void     far receive_esr(void)
{
  _AX = _ES;
  _DS = _AX;
  completed_ecb_ptr = MK_FP(_ES,_SI);
  event_completed_flag = TRUE;
}
```

ซอกเก็ต

สิ่งแรกที่สำคัญในการเขียนโปรแกรมโดยใช้ IPX ก็คือการเปิดซอกเก็ต และอาจเป็นไปได้ว่า ต้องใช้หลายซอกเก็ต ขึ้นอยู่กับแต่ละงาน เราส่งข้อมูลผ่านซอกเก็ตไปยังปลายทาง และเราก็รับข้อมูลทางซอกเก็ตเช่นกัน การใช้ซอกเก็ตต้องระวัง อย่าใช้ซอกเก็ตเดียวกับที่เน็ตแวร์ใช้อยู่ และต้องกำหนดซอกเก็ตในลักษณะที่โปรแกรมของเรา สามารถรู้เบอร์ของซอกเก็ตปลายทางได้ง่ายเมื่อต้องการส่งข้อมูล อีกเรื่องหนึ่งก็คือ เน็ตแวร์กำหนดซอกเก็ต 2 ไบต์ กลับกันโดยเน็ตแวร์ให้ไบต์ที่มีนัยสำคัญสูงสุดมาก่อนหรืออยู่ทางซ้าย ซึ่งจะตรงข้ามกับ ลักษณะการเก็บ

ค่าของรีจีสเตอร์ในซีพียู บางคนใช้วิธีการง่ายๆแก้ปัญหา คือใช้เลขที่เหมือนกันทั้ง 2 ไบต์ เช่น 3434 หรือ 5656 เพื่อว่า ถ้าหากสลับกันแล้ว ก็ไม่แตกต่างอะไร

ซอกเก็ตมีทั้งแบบอายุสั้นและอายุยาว การปิดซอกเก็ตทั้งสองแบบ ใช้คำสั่ง IPX_Close_Socket ซอกเก็ตแบบอายุสั้นจะถูกปิดอัตโนมัติ เมื่อโปรแกรมเสร็จจากการทำงาน แต่ถ้าเป็นซอกเก็ตแบบอายุยาวเราต้องสั่งปิดเอง ตัวอย่างของการใช้งานซอกเก็ตแบบอายุยาว เช่น โปรแกรมแบบ Resident Program

การเปิดซอกเก็ตทำได้โดยใส่เบอร์ของซอกเก็ตลงใน รีจิสเตอร์ DX และเซ็ตค่าให้รีจีสเตอร์ AL ตามที่ต้องการว่าจะใช้ซอกเก็ตแบบอายุสั้นหรืออายุยาว ถ้าต้องการใช้แบบอายุสั้นให้ใช้เป็น 0 ถ้าต้องการแบบอายุยาว ให้ใช้ค่าเป็น FF ฟังก์ชั่นที่ใช้ในการเปิดซอกเก็ต คือฟังก์ชั่นหมายเลข 0 ซึ่งเราต้องใส่หมายเลข ของฟังก์ชั่นลงในรีจีสเตอร์ BX ตัวอย่างต่อไปนี้เป็นการเปิดซอกเก็ต โดยใช้เทอร์โบซี

```
int   open_socket(unsigned int socket)
{
  _DX = socket;
  _BX = 0x0000;
  _AL = 0X00;
  ipx_spx();
  _AH = 0;
  return _AX;
}
```

สำหรับการปิดซอกเก็ต ให้ใส่เบอร์ของซอกเก็ต ลงในรีจีสเตอร์ DX และเรียก IPX โดยใช้หมายเลขฟังก์ชั่นใน BX เป็น 1 ถ้ามีปฏิบัติการส่งหรือรับข้อมูล ผ่านทางซอกเก็ตนั้นที่ยังทำค้างอยู่ ปฏิบัติการนั้นก็จะถูกยกเลิก ต่อไปนี้เป็นฟังก์ชั่นที่ทำหน้าที่ปิดซอกเก็ต

```
void   close_socket(unsigned int socket)
{
  _BX = 0x0001;
  _DX = socket;
  ipx_spx();
}
```

การรับข้อมูล

การรับข้อมูลใน IPX ทำได้โดยการใช้ฟังก์ชั่นเบอร์ 4 และให้คู่รีจิสเตอร์ ES:SI ชี้ไปยัง ECB และจึงเรียก IPX

```
void   ipx_listen_for_packet(sturct ECB *ecb_ptr)
{
   _ES = FP_SEG((void far *) ecb_ptr);
   _SI  = FP_OFF((void far *) ecb_ptr);
   _BX = 0x0004;
   ipx_spx();
}
```

การเตรียมรับข้อมูลให้ IPX เราจะต้องใส่ค่าใน ECB และจึงผ่านค่าให้ IPX สำหรับการเติมค่าในโครงสร้าง ECB จะต้องบอก IPX ว่าซอกเก็ตเบอร์ไหนที่เรากำลังรอรับข้อมูลอยู่ (Listening) และต้องบอกด้วยว่า ถ้ามีข้อมูลเข้ามาจะให้เก็บไว้ที่พื้นที่ใด ข้อมูลที่เข้ามา จะประกอบด้วยสองส่วน คือ ส่วนหัวของแพ็กเก็ต (Packet Header) และส่วนที่เป็นข้อมูล ดังนั้นเราจะต้องเตรียมพื้นที่ไว้สองที่ ในตัวอย่างต่อไปนี้ ตัวแปร packet_count ถูกเซ็ตค่าเป็น 2 เพื่อบอก IPX ว่ามี 2 พื้นที่ แอดเดรสและความยาวของ packet[0] หมายถึงพื้นที่สำหรับส่วนหัวของแพ็กเก็ต แอดแดรสและความยาวของ packet[1] หมายถึงพื้นที่สำหรับข้อมูล (Data Buffer) เมื่อเตรียม ECB เสร็จแล้วก็สามารถเรียกใช้ IPX ได้ และเมื่อ IPX ทำงานเสร็จค่าของ completion_code ใน ECB จะเป็นตัวบอกว่า ทำงานเสร็จหรือไม่

การส่งข้อมูล

ในการรับข้อมูล เราบอกเพียงว่าเมื่อรับข้อมูลมาแล้ว ให้เอาไปไว้ที่ไหน และซอกเก็ตเบอร์อะไร ที่เรากำลังรอรับข้อมูลอยู่ สำหรับการส่งข้อมูล เราต้องบอก IPX มากกว่านั้น

-ที่อยู่ของข้อมูลที่กำลังจะส่ง รวมทั้งพื้นที่สำหรับส่วนหัวของ IPX ด้วย

-เบอร์ของซอกเก็ตที่จะใช้ส่งข้อมูล

-แอดเดรสของ บริดจ์ (Bridge) ที่จะเป็นทางผ่าน (ถ้ามี)

-ประเภทของแพ็กเก็ต ใช้ค่า 4 เสมอถ้าเป็น IPX

-แอดเดรสของผู้รับ ประกอบด้วย แอดเดรสของเครื่อง หมายเลขของเครือข่าย และหมายเลขของซอกเก็ตที่รอรับอยู่

ฟิลด์ Immediate_Address ของ ECB จะประกอบด้วย แอดเดรสของบริดจ์ ซึ่งจะทำหน้าที่ส่งผ่านข้อมูล ถ้าต้องการรู้แอดเดรสของบริดจ์ ก็สามารถใช้ฟังก์ชั่น Get_Local_Target ของ IPX ทำการตรวจสอบได้ Get_Local_Target จะคืนค่าแอดเดรสของ บริดจ์มาให้ และถ้าไม่มีบริดจ์ในระหว่างทางเลย IPX ก็จะคืนค่าของแอดเดรสปลายทางมาให้ ไม่ว่าจะมีบริดจ์หรือไม่ก็ตาม ค่าแอดเดรสที่คืนมา จะอยู่ในฟิลด์ immediate_address เสมอ

ฟังก์ชั่น send() ต่อไปนี้ แสดงให้เห็นการส่งข้อมูลโดยใช้IPX พารามิเตอร์ของฟังก์ชั่นคือ แอดเดรสปลายทาง (dest_network , dest_node และ dest_socket) และข้อมูลที่จะส่ง (packet_ptr และ packet_len) ฟังก์ชั่นsend() จะใส่หมายเลขของซอกเก็ต ที่ใช้ส่ง ลงใน ECB แล้วจึงเรียกใช้ฟังก์ชั่น Get_Local_Target() เพื่อเซ็ตค่าของฟิลด์ immediate_address หลังจากนั้นจึงใส่ค่าแอดเดรส และความยาวของบัฟเฟอร์สำหรับส่วนหัวของ IPX แพ็กเก็ตและส่วนข้อมูลของ IPX แพ็กเก็ตลงใน ECB

ส่วนหัวของแพ็กเก็ตก็คือ 30 ไบต์แรกที่ส่งออกไป ฟังก์ชั่น send() จะใส่แอดเดรสของเครื่องปลายทางลงในส่วนหัวนี้ จากนั้นก็เซ็ตค่าประเภทของแพ็กเก็ตให้เป็น 4 แล้วจึงเรียก IPX เพื่อส่งข้อมูล

```
struct ECB send_ecb;
struct IPXHEADER send_header;
void send(char *dest_network,
          char *dest_node,
          int dest_socket,
          void *packet_ptr,
          int packet_len
{
  int i;
  memset(&send_ecb,0,sizeof(struct ECB));
  send_ecb.socket_number = our_socket;
  i = get_local_target(dest_network,
      dest_node,
      dest_socket,
      send_ecb.immediate_address);
```

```
    if(i != 0) return;
    send_ecb.packet_count = 2;
    send_ecb.packet[0].address = &send_header;
    send_ecb.packet[0[.length = sizeof(struct IPXHEADER);
    send_ecb.packet[1].address = packet_ptr;
    send_ecb.packet[1].length = packet_len;
    send_header.packet_type = 4;
    memcpy(send_header.dest_network_number,
    dest_network,4);
    memcpy(send_header.dest_network_node,
    dest_node,6);
    send_header.dest_network_socket = dest_socket;
    ipx_send_packet(&send_ecb);
}
```

ภาคผนวก ข

การเขียนโปรแกรมเกี่ยวกับเสียง

ไดรฟ์เวอร์เสียง ซึ่งสามารถใช้ในการโปรแกรม การ์ดซาวด์บลาสเตอร์ จะมีให้เลือกใช้ได้ 2 แบบคือ

-CT-VOICE.DRV เป็นไดรฟ์เวอร์เสียง ที่ทำงานกับหน่วยความจำ คือ จะเล่นเสียงจากหน่วยความจำ หรือเก็บข้อมูลเสียง ลงหน่วยความจำ.

-CT-VDISK.DRV เป็นไดรฟ์เวอร์เสียงที่ทำงานกับดิสต์ ซึ่งจะใช้งานในกรณีที่ต้องการเล่น หรือบันทึกแฟ้มเสียง ที่มีขนาดใหญ่กว่าหน่วยความจำที่มีอยู่.

ไดร์ฟเวอร์ดังกล่าว จะทำงานกับรูปแบบของข้อมูลแบบ VOC File ซึ่งวิทยานิพนธ์ฉบับนี้ใช้รูปแบบข้อมูลแบบนี้และทำงานกับหน่วยความจำเท่านั้น ไม่มีการเก็บข้อมูลลงดิสค์ แต่หากใครต้องการใช้ข้อมูลในรูปแบบ WAV File ซาวด์บลาสเตอร์ ก็มีไดร์ฟเวอร์ให้ใช้ได้คือ

-CT-MEM.DRV เป็นไดรฟ์เวอร์ ซึ่งจะเก็บ และเล่นเสียง โดยทำงานกับหน่วยความจำ.

-CT-WDSK.DRV เป็นไดรฟ์เวอร์ซึ่งจะเก็บ และเล่นเสียง โดยทำงานกับดิสค์.

การใช้ไดรฟ์เวอร์เสียงมีขั้นตอนดังต่อไปนี้

- จัดสรรหน่วยความจำ เพื่อสร้างบัฟเฟอร์ โดยให้จุดเริ่มต้นอยู่ที่ออฟเซ็ตศูนย์ ของเซ็กเมนต์.
- โหลดไดรฟ์เวอร์เสียงเข้าสู่บัฟเฟอร์.
- เรียกใช้ฟังก์ชั่นของไดรฟ์เวอร์ ซึ่งทำได้สองวิธี คือ ใช้ไลบราลีซึ่งได้มา กับชุดพัฒนาโปรแกรม (Developer kit) หรือ โดยการเรียกไปยัง ออฟเซ็ตศูนย์ของหน่วยความจำเซ็กเมนต์ที่มีไดรฟ์เวอร์เสียงโหลดอยู่.

เราสามารถตรวจสอบว่า ไดรฟ์เวอร์เสียงถูกโหลดอย่างถูกต้องหรือไม่ โดยการเช็คที่ออฟเซ็ตที่สาม ของเซ็กเมนต์ ซึ่งจะต้องมีข้อความว่า Ct-Voice ปรากฏอยู่

ในการที่จะใช้ไดรฟ์เวอร์อย่างเหมาะสม เราจะต้องทำการกำหนดตัวแปร unsigned int ขึ้นมาหนึ่งตัวมีชื่อว่า ct_voice_status ซึ่งโปรแกรมที่สร้าง จะได้ใช้ตัวแปรนี้ เพื่ออ่านสถานะของไดรฟ์เวอร์ เราสามารถกำหนดแอดเดรสของตัวแปรนี้ให้แก่ไดรฟ์เวอร์ได้ โดยใช้ฟังก์ชั่นหมายเลข 5.

การใช้เรียกแบบ FAR CALL (เรียกแบบข้ามเซ็กเมนต์ได้) สำหรับฟังก์ชั่นทั่วไป (ยกเว้นฟังก์ชั่นพิเศษบางตัว) เราต้องใส่หมายเลขของฟังก์ชั่นลงในรีจิสเตอร์ BX ส่วนรีจีสเตอร์อื่นๆ ยกเว้น AX และ DX จะถูกเรียกค่ากลับคืนมา หลังจากการทำงานของฟังก์ชั่นนั้นๆเสร็จสิ้นลง ตารางต่อไปนี้เป็นการสรุปฟังก์ชันต่างๆ ส่วนรายละเอียดของแต่ละฟังก์ชั่นนั้น จะได้อธิบายเฉพาะฟังก์ชั่น ที่เกี่ยวข้องกับการพัฒนาโปรแกรมเท่านั้น

หมายเลข	ชื่อ
0	Get Driver Version
1	Set Base I/O Address
2	Set DMA Interrupt
3	Initialize Driver
4	Turn DAC Speaker ON/OFF
5	Set Status Word Address
6	Start Voice Output
7	Start Voice Input
8	Stop Voice I/O
9	Terminate Driver
10	Pause Voice Output
11	Continue Voice Output
12	Break Voice Output Loop
13	Set User-Defined Trap
14	Start Voice Output from Extended Memory
15	Start Voice Input to Extended Memory
16	Set Recording Mode (SBPRO Only)
17	Set Recording Source (SBPRO Only)
18	Set Recording Filter (SBPRO Only)
19	Set DMA Channel(SBPRO Only)
20	Get Card Type
21	Reserved
22	Filter ON/OFF(SBPRO Only)

ตาราง ข.1 ฟังก์ชันที่การ์ดซาวนด์บลาสเตอร์มีให้ใช้ได้

23-25	Reserved
26	Get Voice Sampling Rate
27	Read Filter Status (SBPRO Only)
28*	Get Environment Settings
29*	Get Parameter
30*	Set DMA Buffer
31*	Set I/O Parameter
32*	Get I/O Paramater
33*	Input into Conventional Memory
34*	Input into Extended Memory
35*	Output from Conventional Memory
36*	Output from Extenden Memory
37*	Stop Voice I/O
38*	Pause Voice Output
39*	Continue Voice Output
40*	Break Voice Output Loop

ตาราง ข.1 (ต่อ)

ฟังก์ชั่น 3 Initialize Driver

ทำหน้าที่ Initialize ไดรฟ์เวอร์ไฟล์ และตรวจสอบ การ์ดซาวด์บลาสเตอร์ ว่าได้รับการติดตั้งอย่างเหมาะสมหรือไม่ ฟังก์ชั่นนี้จะต้องเรียกก่อนฟังก์ชั่นอื่นๆ ทั้งหมด ยกเว้นเบอร์ 19

ค่าที่ต้องใส่ BX = 3

ค่าที่คืนมา AX = 0 ถ้าไดรฟ์เวอร์ถูก Initialize อย่างถูกต้อง

1 ไดรฟ์เวอร์คนละเวอร์ชั่น

2 การอ่านหรือเขียน อินพุท/เอาต์พุท มีการผิดพลาด

3 การขัดจังหวะของ DMA มีการผิดพลาด

หลังจากการ Initialize ผ่านแล้ว จะเป็นการเปิดสวิตซ์ให้เอาต์พุทผ่านไปยังลำโพงได้

ฟังก์ชั่น 6 Start Voice Output

ฟังก์ชั่นนี้จะทำการ ส่งข้อมูลที่อยู่ในรูปของ VOC ให้ออกที่ลำโพง เมื่อฟังก์ชั่นนี้ถูกเรียก ค่าของ ct_voice_status จะถูกเซ็ตให้เป็น OFFFFH และการควบคุม จะถูกส่งกลับมายังแอพพลิเคชั่นโปรแกรมทันที

ค่าที่ต้องใส่ BX = 6

ES:DI = แอตเครสของเอาต์พุตบัฟเฟอร์

ค่าที่คืนมา AX = 0 ถ้าสำเร็จ

ไม่เท่ากับ 0 ถ้าล้มเหลว

รีจีสเตอร์ ES:DI จะต้องชี้ไปที่ จุดเริ่มต้นของ ข้อมูลจริง (ไม่ใช่จุดเริ่มต้นของไฟล์เสียง) ถ้าต้องการตรวจสอบ ว่าการถ่ายโอนข้อมูลเสร็จหรือยัง ให้เช็คที่ตัวแปร ct_voice_status ซึ่งจะมีค่าเป็นไปได้ 3 แบบ ตอนเริ่มต้น ค่าจะเป็น OFFFFH ต่อมา อาจเปลี่ยนเป็นค่าอื่นก็ได้ที่ไม่ใช่ศูนย์ จนกระทั่ง ข้อมูลถูกถ่ายโอน จนสุดบัฟเฟอร์ หรือ โพรเซสถูกสั่งหยุดด้วยฟังก์ชั่นเบอร์ 8 ค่าจะเปลี่ยนเป็นศูนย์

ฟังก์ชั่น 7 Start Voice Input

สั่งเริ่มการทำการถ่ายโอนข้อมูลสำหรับการรับเสียงเข้า เมื่อฟังก์ชั่นนี้ถูกเรียกค่าของ ct_voice_status จะถูกเซ็ตเป็น OFFFFH และคืนการควบคุมกลับสู่แอพลิเคชั่นโปรแกรม

ค่าที่ต้องใส่ BX = 7

AX = อัตราการชักตัวอย่างมีหน่วยเป็น เฮิตซ์

DX:CX = ความยาวของ อินพุตบัฟเฟอร์มีหน่วยเป็นไบต์

ES:DI = ตำแหน่งของอินพุตบัฟเฟอร์

ค่าที่คืนมา AX = 0 ถ้าสำเร็จ

ไม่เท่ากับ 0 ถ้าล้มเหลว

สำหรับอัตราการชักตัวอย่างจะใช้ค่าแตกต่างกันไปขึ้นอยู่กับรุ่นของการ์ดเสียงที่ใช้ดังนี้

การ์ด	อัตราการชักตัวอย่าง
SB PRO	4000-44100 เฮิรตซ์ โมโน 22050 หรือ 44100 เฮิรตซ์ สเตริโอ
SB V 2.0	4000-15000 เฮิรตซ์ โมโน
SB V 1.5	4000-13000 เฮิรตซ์ โมโน

ตาราง ข.2 อัตราการชักตัวอย่างที่ใช้ได้กับการ์ดซาวนด์บลาสเตอร์แต่ละรุ่น

การอินพุตเสียงนี้ จะหยุดก็ต่อเมื่อ อินพุตบัฟเฟอร์เต็ม หรือเมื่อโปรแกรมสั่งหยุดโดยการเรียกฟังก์ชั่นเบอร์ 8 ถ้าเกิดเหตุการณ์เหล่านี้ ตัวแปร ct_voice_status จะถูกเซ็ตเป็นศูนย์

ค่าการชักตัวอย่าง แบบสเตอริโอ สำหรับ SB PRO มีได้แค่ 2 ค่าเท่านั้นคือ 11025 เฮิรตซ์ หรือ 22050 เฮิรตซ์ การใส่ค่าการชักตัวอย่าง แบบสเตอริโอทำได้โดย กำหนดค่าให้กับรีจีสเตอร์ AX สองเท่าของอัตราของชักตัวอย่าง ที่ต้องการ ถ้าต้องการตัวอย่าง 11025 เฮิรตซ์ ให้ใช้ค่า AX = 22050 ถ้าต้องการอัตราการชักตัวอย่าง 22050 เฮิรตซ์ ให้ใช้ค่า AX เป็น 44100

ฟังก์ชั่น 8 Stop Voice I/O

ทำการหยุดโพรเซสใดๆ ที่เกี่ยวกับการอินพุต หรือเอาต์พุตเสียง และเซ็ตค่า ct_voice_status เป็น 0

ค่าที่ต้องใส่ BX = 8

ค่าที่คืนมา -

ฟังก์ชั่น 17 Set Recording Source

ทำหน้าที่เลือกอุปกรณ์กำเนิดเสียงเพื่อทำการบันทึก

ค่าที่ต้องใส่ BX = 17

AX = 1 CD

= 2 ไมโครโฟน

= 3 LINE - IN

ค่าที่คืนมา AX = เบอร์ของอุปกรณ์กำเนิดเสียงที่ใช้ก่อนหน้านี้

ประวัติผู้เขียน

นายกิตติวุฒิ ชูโชติ เกิดวันที่ 9 มีนาคม พ.ศ.2511 ที่อำเภอบางกอกน้อย จังหวัดกรุงเทพมหานคร สำเร็จการศึกษา ระดับอนุปริญญา ในสาขาอีเลคทรอนิคซ์ จากศูนย์ฝึกการบินพลเรือน ในปี พ.ศ.2533 สำเร็จการศึกษา ระดับปริญญาตรี สาขาวิทยาศาสตร์คอมพิวเตอร์ คณะวิทยาศาสตร์ มหาวิทยาลัยมหิดล ในปีการศึกษา 2535 จากนั้นเข้าศึกษาต่อ ในหลักสูตรวิทยาศาสตร์มหาบัณฑิต สาขาวิทยาศาสตร์คอมพิวเตอร์ จุฬาลงกรณ์มหาวิทยาลัยเมื่อพ.ศ.2536 ปัจจุบันทำงานอยู่ที่ สำนักงานซีต้าเทเลคอม ในตำแหน่งพนักงานฝึกอบรม