Sena Technologies, Inc.

HelloDevice STS 시리즈

사용자 커스터마이제이션 가이드

버전 **1.1.0**

Sena Technologies, Inc.

저작권

상표 정보

기술 지원

㈜세나테크놀로지
서울시 서초구 양재동 210 (우)137-130
Tel: (02) 573-5422
Fax: (02) 573-7710
E-Mail: support@sena.com
Website: http://www.sena.com

Revision history

Revision	Date	Name	Description
V0.0.1	**2003-09-09**	**H. Yeom**	Initial Draft
V1.0.0	**2003-09-22**	**H. Yeom**	Initial Release
V1.1.0	**2004-01-12**	**H. Yeom**	GDB 지원 및 샘플 필터 프로그램 추가

목 차

1. 개요.....5
2. SDK (소프트웨어 개발 키트).....6
3. 사용자 프로그램을 구축하는 방법.....7
3.1. 준비.....7
3.2. 코딩.....7
3.3. 파일 업로드하기.....7
3.4. 구축 및 실행.....7
4. 사용자 웹 커스터마이징.....8
4.1. HTML 파일.....8
4.2. CGI 파일.....8
4.3. 자바 애플릿과 그 외 파일들.....10
5. 사용자 필터 커스터마이징.....10
5.1. 필터 프로그램 이해하기.....10
5.2. 필터 프로그램 구축하기.....11
5.3. 필터 샘플.....12
6. GDB로 디버깅하기.....13
6.1. GDB을 이용한 샘플 필터 프로그램 디버깅.....13

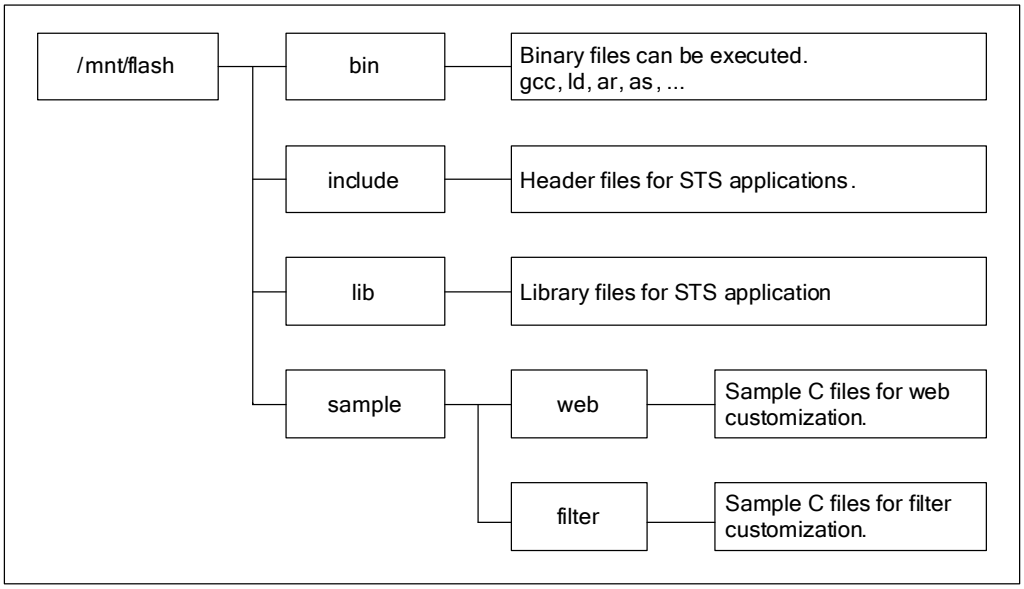
1. 개요

HelloDevice STS 시리즈는 프로그래밍 가능한 시리얼-이더넷 통신 게이트웨이로서 RS-232 기반의 시리얼 장치에 사용되며 다양한 커스터마이징 옵션을 가지고 있습니다.
㈜세나테크놀로지는 HelloDevice STS 시리즈용 사용자 어플리케이션 개발을 위해 사용하기 쉬운 소프트웨어 개발 키트(SDK) 환경을 제공합니다. 사용자는 웹 관리 인터페이스를 커스터마이징 할 수 있으며, 프로그램된 동적 웹 페이지를 웹 메뉴로 통합할 수 있습니다.
또한, 사용자는 시리얼 장치에서 소켓으로, 혹은 소켓에서 시리얼 장치로 통하는 데이터 흐름을 관리할 수 있습니다. 사용자 설정 필터 프로그램은 FIFO 방식을 사용하여 시리얼 포트와 소켓을 읽고 쓸 수 있는(read/write) 다른 프로그램들과 통신하며, 이를 통해 사용자는 시리얼 포트나 소켓과 관련된 프로그래밍 없이도 시리얼 데이터를 쉽게 다룰 수 있게 됩니다.
HelloDevice STS 시리즈는 임베디드 Linux OS에서 실행되며, UNIX / Linux 명령어를 이용하여 코멘드-라인 인터페이스 또는 준비된 여러 개의 스크립트들을 이용한 프로그래밍이 가능합니다.

2. SDK (소프트웨어 개발 키트)

사용자가 어플리케이션 코드를 생성하려면 HelloDevice STS 시리즈용 SDK가 필요합니다. HelloDevice STS 시리즈 SDK는 PC CF 카드의 형태 또는 웹 다운로드의 형태로 제공됩니다. HelloDevice STS 시리즈용 SDK가 필요하시면 ㈜세나테크놀로지 기술 지원 센터에 연락하십시오. SDK는 컴파일러, 링커, 라이브러리 파일, 헤더 파일 및 C 파일 샘플 파일들을 포함합니다.

SDK의 디렉토리 구조는 다음과 같습니다.



3. 사용자 프로그램을 구축하는 방법

3.1. 준비

커스터마이징을 위한 개발환경을 준비하기 위해 다음 과정을 따릅니다.

1) CF 카드 형태의 HelloDevice STS 시리즈 용 SDK를 준비합니다.

2) STS 장치의 PCMCIA (PC 카드) 슬롯에 SDK를 삽입합니다.

3) 설정 메뉴에서 PC 카드를 찾고, 이를 저장한 뒤 적용합니다.

3.2. 코딩

C 언어로 된 소스 파일을 제작합니다. 소스 파일은 HelloDevice STS 시리즈 CLI 에서 편집할 수 있지만, 사용이 불편할 경우 PC 에서도 할 수 있습니다. Linux 라이브러리는 /mnt/flash/lib 디렉토리에 위치해 있습니다.

3.3. 파일 업로드하기

HelloDevice STS 시리즈 CLI에서 파일을 편집할 경우에는 이 부분을 읽지 않으셔도 됩니다. PC 에서 소스 파일을 작성한 후, 소스 파일을 컴파일하려면, 반드시 HelloDevice STS 시리즈 장치에 파일을 업로드해야 합니다. 다음의 세 가지 방법으로 파일을 업로드 할 수 있습니다.

- SCP
- FTP
- 설정 메뉴

더 자세한 정보가 필요하시면 사용자 가이드를 참조하십시오.

3.4. 구축 및 실행

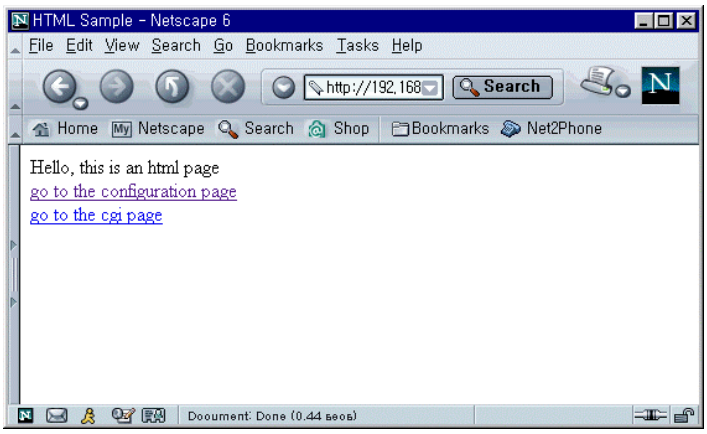
프로그램을 다음과 같이 컴파일 및 링크한 후 실행합니다. (아래 예는 proga.c와 progb.c라는 두개의 C 소스 파일로 구성된 *prog* 이라는 프로그램이라고 가정한 것입니다.)

```
# gcc –o  prog proga.c progb.c
# ./prog
```

4. 사용자 웹 커스터마이징

4.1. HTML 파일

HelloDevice STS 시리즈에 사용자가 만든 HTML 파일들을 추가함으로써 웹 관리 인터페이스를 사용자가 커스터마이징할 수 있습니다. 모든 HTML 파일들은 반드시 `/usr2/usrweb` 디렉토리 내에 있어야 합니다. `index.html`이라는 HTML 파일 샘플을 `/usr2/usrweb` 으로 복사하면, 아래와 같은 HTML 페이지를 볼 수 있습니다.



4.2. CGI 파일

4.2.1 CGI 파일 구축하기

CGI 파일을 구축하는 절차는 다음과 같습니다.

CGI 소스 파일의 샘플은 `/mnt/flash/sample/web/cgi/shell.c` 입니다.

1 단계. CGI 파일을 구축합니다.

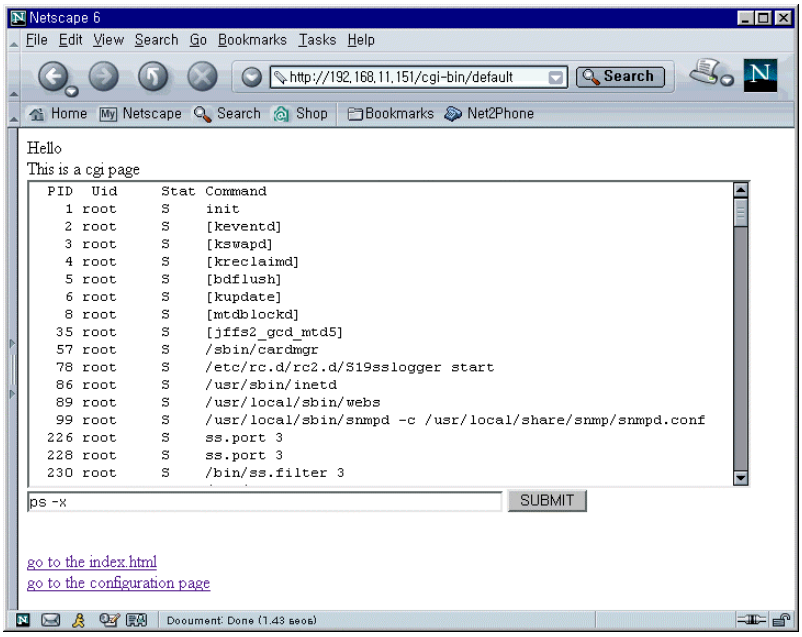
```
# cd /mnt/flash/sample/web/cgi
# make
```

2 단계. CGI 파일을 복사합니다.

```
# cp shell.cgi /usr2/cgi-bin/
```

3 단계. 웹 브라우져를 실행 하고, STS400/800/1600에 접속하여 로그인 합니다.

4 단계. 웹 브라우져에서 http://192.168.1.2/cgi-bin/shell.cgi로 이동합니다. (HelloDevice STS 시리즈의 IP 주소는 192.168.1.2라고 가정합니다.)



4.2.2 CGI 파일 사용하기

CGI 파일은 반드시 `/usr2/cgi-bin` 디렉토리 내에 있어야 합니다. 만약 default 라는 CGI 파일이 있으면, 그 파일은 커스텀 페이지의 시작 페이지가 될 수 있습니다.

아래는 `Shell.c` 를 만들기 위한 Makefile 입니다.

```
CC = gcc
BIN = shell.cgi
OBJS = shell.o util_cgi.o
LDFLAG = -L/mnt/flash/lib

BIN : $(OBJS)
	$(CC) -o $(BIN) $(OBJS) $(LDFLAG)
c.o :
	$(CC) -c $<
all : $(BIN)
clean :
	rm -f $(BIN) $(OBJS)
```

`util_cgi.h`와 `util_cgi.c` 는 Appendix A에 첨부하였습니다. 이러한 CGI 프로그램은 반드시 `/usr2/cgi-bin` 디렉토리 내에 있어야 합니다.

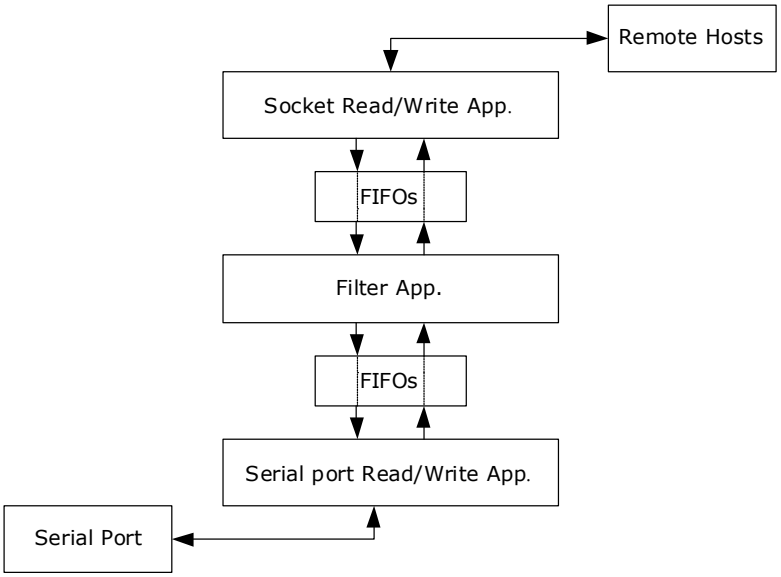
4.3. 자바 애플릿과 그 외 파일들

자바 애플릿 역시 사용할 수 있습니다. CGI 프로그램을 제외한 애플릿 파일과 모든 웹 파일들은 반드시 /usr2/usrweb 디렉토리 내에 있어야 합니다.

5. 사용자 필터 커스터마이징

5.1. 필터 프로그램 이해하기

HelloDevice STS 시리즈는 FIFO 방식을 사용하여 프로세스간 통신을 합니다. 사용자는 시리얼 장치에서 소켓으로, 혹은 소켓에서 시리얼 장치로 통하는 데이터 흐름을 관리하기 위해서 프로그램을 삽입할 수 있습니다.
사용자가 설정한 필터 프로그램은 FIFO 방식을 사용하여 시리얼 포트와 소켓을 읽고 쓰는(read/write) 다른 프로그램들과 통신합니다.



사용자가 설정한 필터는 시리얼 포트 데이터를 스트리밍하는 FIFO를 읽고, 데이터를 처리한 뒤 처리된 데이터를 소켓으로 전송되는 FIFO에 기록합니다. 소켓에서 시리얼 포트로 가는 데이터의 흐름도 같은 방식으로 처리됩니다. 두 경우 모두 반드시 다음의 조건을 만족시켜야 합니다.

1) 시리얼 포트로 가는 데이터 흐름을 /tmp/port_fifos/port***X***_f2s라는 FIFO에 기록합니다 (단, *X*는 1을 기준으로 한 포트 인덱스).
2) 시리얼 포트로 가는 데이터 흐름을 /tmp/port_fifos/port*X*_f2e라는 FIFO에 기록합니다.
3) 시리얼 포트로부터 /tmp/port_fifos/port*X*_s2f라는 FIFO에 들어오는 데이터 흐름을 동일한 FIFO로부터 읽습니다.
4) 소켓으로부터 /tmp/port_fifos/port*X*_e2f라는 FIFO에 들어오는 데이터 흐름을 동일한 FIFO로부터 읽습니다.
5) 종료하지 않는 한, 실행되고 있는 네 개의 FIFO를 닫지 마십시오.
6) 한 개 이상의 변수가 있어야 하며, 첫번째 변수는 포트 번호로 지정해야 합니다.
7) /var/run/port*X*_filter.pid 라는 파일에 PID(Process ID)를 기록합니다. (이 PID 파일은 필터 어플리케이션을 종료하는데 쓰이지만, 포트가 비활성화(disable) 상태일 경우에만 종료합니다.)
8) SIGTERM 신호를 받으면 종료됩니다.

5.2. 필터 프로그램 구축하기

필터 프로그램을 구축하는 절차는 아래와 같습니다.

샘플 프로그램은 /mnt/flash/sample/filter/ 디렉토리에 있습니다.

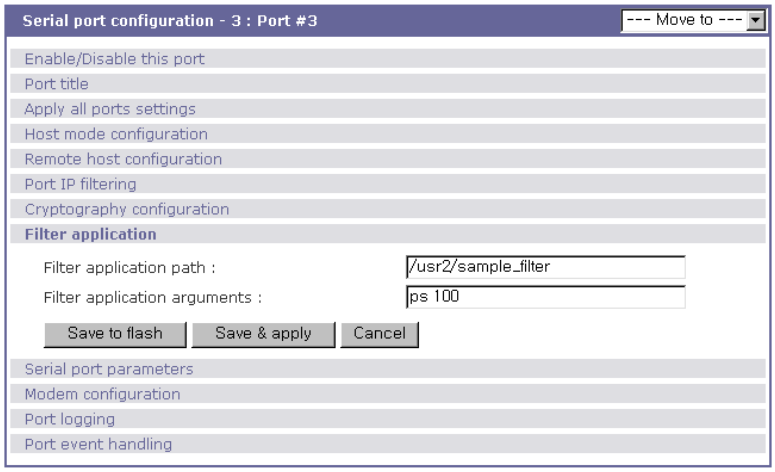
1 단계. 필터 프로그램을 구축합니다.

```
# cd /mnt/flash/sample/filter
# make
```

2 단계. 필터 프로그램 중 하나의 샘플을 사용자 공간으로 복사합니다.

```
# cp data_conversion /usr2/sample_filter
```

3 단계. 필터 어플리케이션 구성을 다음과 같이 설정합니다:



5.3. 필터 샘플

모든 샘플 프로그램은 메인 Thread, 시리얼 포트를 읽는 Thread 및 소켓을 읽는 Thread 등 총 세 개의 Thread로 구성되어있습니다. 메인 Thread는 다른 Thread를 만들고, SIGTERM 신호를 받을 때까지 기다립니다. 메인 Thread는 SIGTERM 신호를 받으면 다른 Thread들을 취소합니다. 시리얼 포트/소켓을 읽는 Thread는 목적에 따라 수정될 수 있습니다.

- empty.c

이 샘플 필터는 시리얼 포트 읽기/쓰기 어플리케이션과 소켓 읽기/쓰기 어플리케이션을 FIFO로 연결합니다. 그러나 데이터를 다루지는 않습니다.

- periodic_command.c

이 샘플 필터는 시리얼 포트 읽기/쓰기 어플리케이션과 소켓 읽기/쓰기 어플리케이션을 FIFO로 연결하며, 시리얼 포트에 주기적으로 명령을 기록합니다.

- data_conversion.c

이 샘플 필터는 프로토콜 변환에 대한 예제입니다.

- data_calibration.c

이 샘플 필터는 주기적으로 들어오는 시리얼 포트 데이터들의 평균을 계산하고(예: 온도계 데이터, 습도계 데이터) 계산 값을 보내주는 예제입니다.

- data_storing.c

이 샘플 필터는 시리얼 포트 데이터를 감시하고(예:온도계 데이터, 습도계 데이터) RAM 디스크 (/tmp 디렉토리 아래)에 저장하는 예제입니다.

- data_event_handling.c

이 샘플 필터는 시리얼 포트 데이터를 감시하고(예:온도계 데이터, 습도계 데이터) SNMP 트랩으로 보내는 예제입니다.

- cq.c

Circular queue 용 유틸리티 루틴.

6. GDB로 디버깅하기

HelloDevice STS 시리즈용 SDK는 GNU GDB 디버거를 지원하므로 사용자 프로그램이 실행되는 동안 그 내부에서 일어나고 있는 일을 사용자가 볼 수 있습니다. (참고: GDB 지원은 SDK v1.1.0 및 후속 버전부터 포함됩니다.)

GDB는 네가지 방법으로 실행 중인 버그를 발견합니다:

- 프로그램을 시작하여 실행에 영향을 미칠 수 있는 것들을 찾아냅니다.□
- 사용자의 프로그램을 특정 조건에서 멈추도록 합니다.
- 사용자의 프로그램이 멈추었을 때, 어떤 일이 일어났는지 조사합니다.□
- 사용자의 프로그램 내용을 변경하여 버그 하나를 고쳤을 때의 결과를 보여주며 또 다른 버그를 찾도록 합니다.

GDB는 쉘 명령 gdb 로 실행됩니다. 한번 시작하면, GDB는 사용자가 ***quit*** 이라는 GDB 명령어를 사용하여 나가도록 지시할 때까지 단말기로부터 명령을 읽습니다. 사용자는 ***help***라는 명령어로 GDB 자체로부터 온라인 상의 도움을 받을 수 있습니다.
사용자는 인수나 옵션 없이 GDB를 구동 시킬 수 있습니다. 그러나 흔히 GDB는 인수로서 실행 가능한 프로그램을 지정하여 한 개 또는 두 개의 인수로 시작합니다.
사용자 프로그램을 GDB로 구동하기 전에, 사용자는 -g 옵션으로 프로그램을 컴파일 해야 합니다.

6.1. GDB을 이용한 샘플 필터 프로그램 디버깅

이 장에서는 GDB를 이용한 샘플 필터 프로그램 디버깅 방법을 단계 별로 설명합니다. 샘플 필터 프로그램을 제대로 디버그 하려면, 사용자는 Web UI나 editconf 같은 설정 도구를 사용하여 시리얼 포트에 대한 각각의 매개변수를 미리 설정해야 합니다. 또한, 사용자가 각각의

포트 기능을 적절한 시점에 일일이 구동할 수 있도록 설정 도구를 사용하여 모든 시리얼 포트를 선택 해제해야 합니다.

1 단계. 샘플 프로그램을 사용자 공간으로 복사하십시오.

```
# cp /mnt/flash/sample/filter/Makefile /usr2/
# cp /mnt/flash/sample/filter/data_conversion.c /usr2/
# cp /mnt/flash/sample/filter/cq.c /usr2/
# cp /mnt/flash/sample/filter/cq.h /usr2/
```

2 단계. 프로그램을 –g 옵션으로 컴파일 하기 위해 Makefile을 수정합니다.

–g 옵션을 CFLAGS 변수에 추가합니다.

```
# cd /usr2/
# vi Makefile
...
//CFLAGS = -pipe
CFLAGS = -pipe –g
...
```

3 단계. 소스 프로그램을 GDB 디버거와 함께 구동 되도록 수정합니다. do_daemon() 및 save_pid(portnum) 기능 호출을 제거합니다.

사용자는 GDB로 디버그 하기 위해 프로그램을 포그라운드(foreground)에서 실행해야 합니다.

```
# vi data_conversion.c
...
int main(int argc, char **argv)
{
     if (argc < 2){
           fprintf(stderr,      "\nUsage:      %s      [portnumber]
[echo|no_echo]\n\n", get_program_name(argv[0]));
           return -1;
     }

     portnum = atoi(argv[1]);
     if (argc>2 && !strcmp(argv[2], "no_echo")) echo_flag = 0;

     (void) signal(SIGTERM, handle_sigterm);
     (void) signal(SIGPIPE, handle_sigterm);

// Remark following two lines to run this program on the foreground
//   do_daemon();
//   save_pid(portnum);

     do_filter();
     close_fifos();

     return 0;
```

```
}
...
```

4 단계. 샘플 필터 프로그램을 컴파일 합니다.

```
# make data_conversion
```

5 단계. 시리얼 포트 데몬(ss.port)과 tcp 소켓 데몬(ss.tcp) 프로그램을 수동으로 시작합니다. ss.port는 시리얼 포트용 데몬 프로그램이며, ss.tcp는 TCP 포트용 데몬 프로그램입니다. 이 두 가지 프로그램들은 STS 장치의 /bin 디렉토리에 위치해 있습니다. 인수 '1'은 1번 포트에서 구동 되고 있음을 의미합니다.

```
# ss.port 1
# ss.tcp 1
# ps –ef
   PID  Uid    Stat Command
    1 root    S    init
    2 root    S    [keventd]
    3 root    S    [kswapd]
    4 root    S    [kreclaimd]
    5 root    S    [bdflush]
    6 root    S    [kupdate]
    8 root    S    [mtdblockd]
   35 root    S    [jffs2_gcd_mtd5]
   57 root    S    /sbin/cardmgr
   80 root    S    dhcpcd eth0
   87 root    S    /etc/rc.d/rc2.d/S19sslogger start
   97 root    S    /usr/sbin/inetd
  100 root    S    /usr/local/sbin/webs
  110   root               S            /usr/local/sbin/snmpd   -c
/usr/local/share/snmp/snmpd.conf
  113 root    S    /bin/linkupchecker -c 1
  118 root    R    /etc/rc.d/rc2.d/S53sts800mand start
  126 root    S    /etc/rc.d/rc2.d/S53sts800mand start
  128 root    S    /usr/sbin/cron
  129 root    S    -bash
  523 root    S    ss.port 1
  524 root    S    ss.port 1
  525 root    S    ss.port 1
  526 root    S    ss.port 1
  527 root    S    ss.port 1
  529 root    S    ss.tcp 1
  530 root    S    ss.tcp 1
  532 root    S    ss.tcp 1
  533 root    S    ss.tcp 1
  534 root    S    ss.tcp 1
  535 root    R    ps -ef
```

6 단계. GDB를 사용하여 샘플 필터 프로그램을 구동합니다.

```
# /mnt/flash/bin/gdb data_conversion
```

7 단계. GDB를 구동한 후, 사용자는 GDB 명령을 이용하여 프로그램을 제어할 수 있습니다. 이 예제에서, 사용자는 먼저 샘플 필터 프로그램의 인수를 다음과 같이 설정해야 합니다.

```
GNU gdb 5.0
Copyright 2000 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and
you are
welcome to change it and/or distribute copies of it under certain
conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for
details.
This GDB was configured as "powerpc-hardhat-linux"...
(gdb) set args 1
```

인수'1'은 샘플 필터 프로그램이 1번 포트에서 구동 되고 있음을 의미합니다.

8 단계. 프로그램 실행을 정지시킬 적절한 위치에 정지 지점을 설정합니다. 예를 들어, 사용자는 정지 지점을 *e2s_thread 함수에 설정하여, TCP 포트로부터 무엇이 들어오는 가를 확인 할 수 있습니다.

```
(gdb) break *e2s_thread
 Breakpoint 1 at 0x10003054: file data_conversion.c, line 255.
```

그 후에 프로그램을 'r' 명령어로 구동합니다.

```
(gdb) r
Starting program: /usr2/data_conversion 1
[New Thread 539 (manager thread)]
[New Thread 538 (initial thread)]
[New Thread 540]
[New Thread 541]
[Switching to Thread 541]

Breakpoint 1, e2s_thread (arg=0x0) at data_conversion.c:255
255    {
(gdb)
```

9 단계. 잠시 후, 프로그램은 *e2s_thread 함수의 시작점에서 정지하게 됩니다. 사용자는 'n'(다음) 명령어를 입력하여 다음 단계를 구동 시킬 수 있습니다.

```
(gdb) n
e2s_thread (arg=0x0) at data_conversion.c:257
257           int nread=0;
(gdb) n
261           pthread_setcancelstate(PTHREAD_CANCEL_ENABLE, NULL);
(gdb) n
262                pthread_setcanceltype(PTHREAD_CANCEL_ASYNCHRONOUS,
NULL);
(gdb) n
...
```

10 단계. 프로그램 실행이 274번째 줄에 도달하면, 사용자는 STS 장치의 1번 시리얼 포트에 TCP를 연결해주고, TeraTerm Pro 나 Hyper Terminal과 같은 단말기 에뮬레이션

프로그램을 사용하여 1번 시리얼 포트에 데이터를 전송합니다. 이렇게 하면, 샘플 필터 프로그램이 TCP 버퍼로부터 데이터를 읽고 다음 단계로 진행할 수 있습니다

```
...
(gdb) n
270             if (f2s_fd < 0) pthread_exit(NULL);
(gdb) n
272             while(!exit_flag) {
(gdb) n
274                     nread = read(e2f_fd, buf, sizeof(buf));
(gdb) n
275                     if (nread<=0) continue;
(gdb)
```

11 단계. 정지 지점을 275번째 줄에 설정하고 표시변수 `buf[0]`를 설정함으로써, 사용자는 TCP포트로부터 오는 데이터가 있을 때에 언제라도 그것을 모니터 할 수 있습니다.

```
(gdb) display buf[0]
1: buf[0] = 97 'a'
(gdb) break 275
Breakpoint 2 at 0x10003160: file data_conversion.c, line 275.
(gdb) n
277                     if (echo_flag) {
1: buf[0] = 97 'a'
(gdb) c
Continuing.

Breakpoint 2, e2s_thread (arg=0x0) at data_conversion.c:275
275                     if (nread<=0) continue;
1: buf[0] = 50 '2'
(gdb) c
Continuing.

Breakpoint 2, e2s_thread (arg=0x0) at data_conversion.c:275
275                     if (nread<=0) continue;
1: buf[0] = 51 '3'
(gdb)
```

12 단계. 사용자는 GDB 디버거를 '***quit***' 명령어를 사용하여 중지시킬 수 있습니다.

```
(gdb) quit
The program is running.  Exit anyway? (y or n) y
```

샘플 필터 프로그램을 다시 구동하려면 `ss.port`와 `ss.tcp` 데몬을 삭제한 후에, 5 단계부터 다시 시작합니다.

```
# killall ss.port
# killall ss.tcp
```

GDB 디버거에 대한 자세한 정보는 GNU 다큐멘테이션 페이지를 참조하십시오. (http://www.gnu.org/software/gdb/documentation/)