

Title	Super Series User Customization Guide
No.	

Rev.	Date	By	History
0.0.1	2003-09-09	H. Yeom	Initial draft
0.0.2	2003-09-16	H. Yeom	

Content

1. Overview	2
2. SDK(Software Development Kit)	3
3. How to build user programs?.....	3
3.1. Preparing	3
3.2. Coding	3
3.3. Uploading Files	3
3.4. Building and Executing	4
4. User Web Customization	4
4.1. HTML Files	4
4.2. CGI files	4
4.3. Java applets and so on	5
5. User Filter Customization	6
5.1. Understanding Filter Programs	6
5.2. Building Filter Program	7
Appendix A. CGI source files	8
A.1 shell.c	8
A.2 cgi_util.h	12
A.3 cgi_util.c	12
A.4 Makefile	15
Appendix B. Sample Filter source files	15
B.1 empty_filter.c	15
B.2 periodic_command_filter.c	18

1. Overview

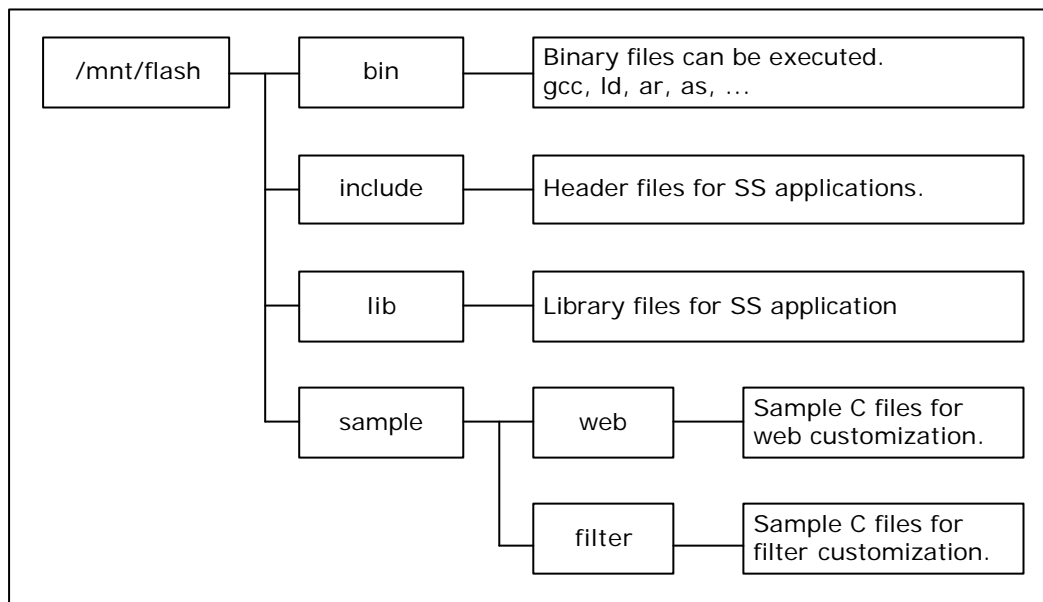
The HelloDevice Super Series is a serial to Ethernet programmable communication gateway for RS-232/422/485 based serial devices with various customization options.

Sena provides easy to use Software Development Kit (SDK) environment to quickly develop custom applications that run on the HelloDevice Super Series. Users can customize the web management interface, and integrate the programmed dynamic web pages to web menu.

In addition, user can manipulate the data stream which routes from serial device to socket and socket to serial device. The user-defined filter program communicates with other programs that are reading/writing serial port and socket by using FIFOs, and so user can easily manipulate the serial data without programming related to serial port and socket. The unit runs the embedded Linux Operating system and facilitates programming using command-line interface or one of several prepared scripts, using UNIX / Linux commands.

2. SDK (Software Development Kit)

To make user's own application code, SDK for SS110/400/800 is needed. SS110/400/800 SDK is provided in the form of PC CF card (Please contact Sena Technical Support to get SDK for SS110/400/800.). The SDK contains compiler, linker, library files, header files and some sample C files. The directory structure of the SDK is as follows.



3. How to build user programs?

3.1. Preparing

Follow the below steps to prepare the development environment for customization.

- 1) Get a SDK for SS110/400/800 in the form of CF card.
- 2) Insert the SDK into the PCMCIA (PC card) slot of the SS device.
- 3) On the configuration menu, probe the PC card and then save and apply it.

3.2. Coding

Write source files in the C language and Makefiles. You can edit the source files on the Super Series CLI but if it is uncomfortable, you can do it on PC or Linux box. Almost all the Linux libraries are located at `/mnt/flash/lib` directory.

3.3. Uploading Files

If you edit the files on the Super Series CLI, skip over this section. To compile the source files, you must have uploaded the files to the SS device. You can upload the files in three ways as follows.

- SCP
- FTP
- Configuration menu

For more detail information, refer to the User Guide.

3.4. Building and Executing

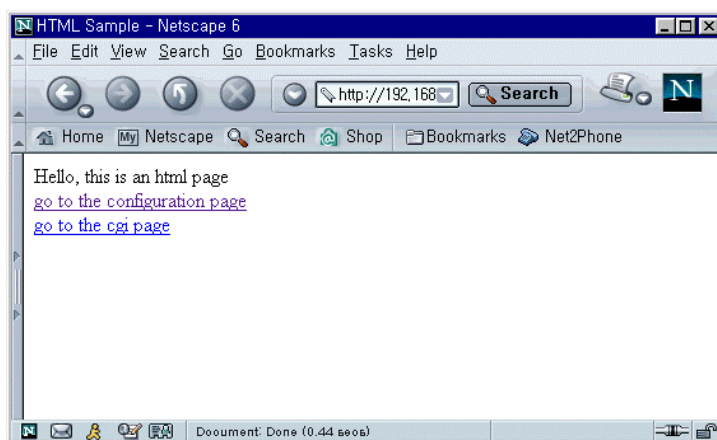
Compile, link and execute the programs as follows. (Imagine, a program named *prog* consists of two C source files *proga.c* and *progb.c*).

```
$ gcc -o prog proga.c progb.c
$ ./prog
```

4. User Web Customization

4.1. HTML Files

You may customize the web management interface by adding your own HTML files in Super Series. All the HTML files must be in the */usr2/usrweb* directory. There is a sample HTML file that is named *index.html*. Copy the file to */usr2/usrweb*, and then you can see the HTML page like below.



4.2. CGI files

4.2.1 Building CGI files

Below is the procedure to build CGI files. Sample CGI source file is located at */mnt/flash/sample/web/cgi/shell.c*,

Step 1. Build the CGI file.

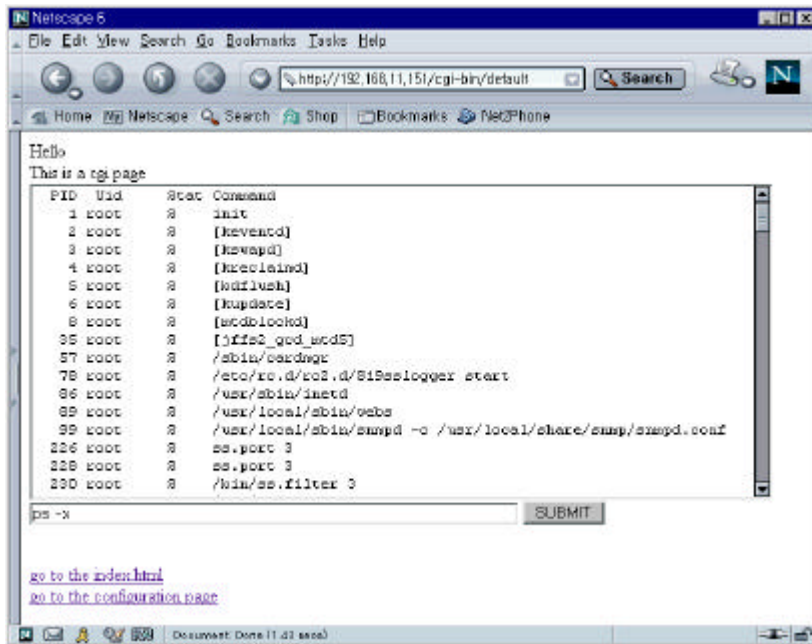
```
$ cd /mnt/flash/sample/web/cgi
$ make
```

Step 2. Copy the CGI file.

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

Step 3. Execute a Web browser, connect to the SS110/400/800, and log in.

Step 4. At the Web browser, go to `http://192.168.1.2/cgi-bin/shell.cgi`. (Suppose that the IP address of the SS110/400/800 is 192.168.1.2)



4.2.2 Using CGI files

The CGI files must be in the `/usr2/cgi-bin` directory. If there is a CGI file named `default`, the file can be an entry page of the custom pages.

Here is a Makefile to make the `shell.c`.

```
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)
```

The `util_cgi.h` and `util_cgi.c` are attached at the Appendix A. These CGI programs must be in the `/usr2/cgi-bin` directory.

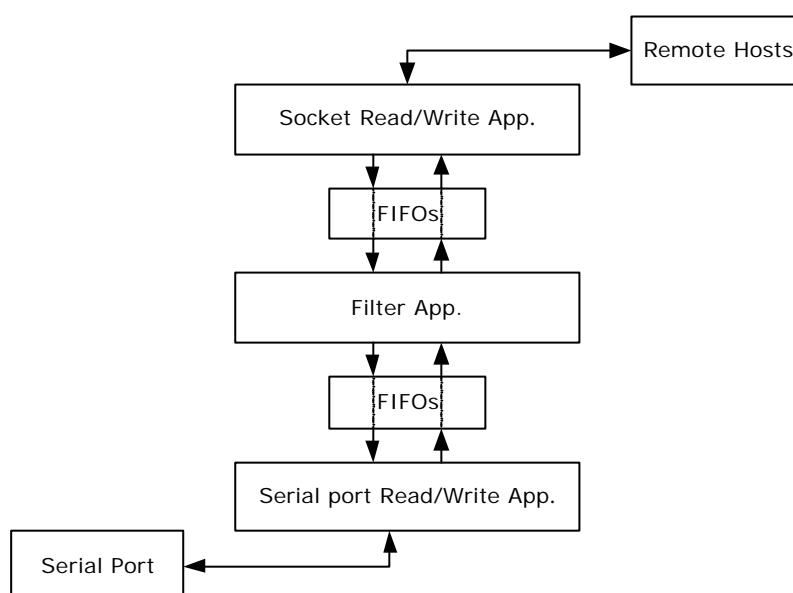
4.3. Java applets and so on

JAVA applets are also available. The applet files and all web files except CGI program must be in the `/usr2/usrweb` directory.

5. User Filter Customization

5.1. Understanding Filter Programs

SS110/400/800 uses FIFO method for inter-process communication. User can insert a program to manipulate the data stream which routes from serial device to socket and socket to serial device. The user-defined filter program communicates with other programs that are reading/writing serial port and socket by using FIFOs.



The user-defined filter reads a FIFO that is streaming the serial port data, manipulate the data and write the manipulated data to a FIFO that is sent to socket. The data stream that goes from socket to serial port can be manipulated in the same way. It must be satisfactory for following qualifications.

- 1) Write the data stream going to serial port to FIFO that is named
`/tmp/port_fifos/portX_f2s` (X is the port index based on 1).
- 2) Write the data stream going to serial port to FIFO that is named
`/tmp/port_fifos/portX_f2e`.
- 3) Read the data stream coming from serial port to FIFO that is named
`/tmp/port_fifos/portX_s2f`.
- 4) Read the data stream coming from socket to FIFO that is named
`/tmp/port_fifos/portX_e2f`.
- 5) Open four FIFOs on being executing and do not close the FIFOs except to be terminated.
- 6) Have more than one arguments and the first argument must indicate the port number.
- 7) Record the PID (Process ID) to the file that is named `/var/run/portX_filter.pid`.

(This PID file is used to terminate the filter application, but it is terminated only in case the port is disabled.)

- 8) Be terminated when it receives the SIGTERM signal.

5.2. Building Filter Program

Below is the procedure to build Filter program. The sample program, which is located at /mnt/flash/sample/filter/periodic_filter.c, is a filter source file that writes a command to serial port, periodically.

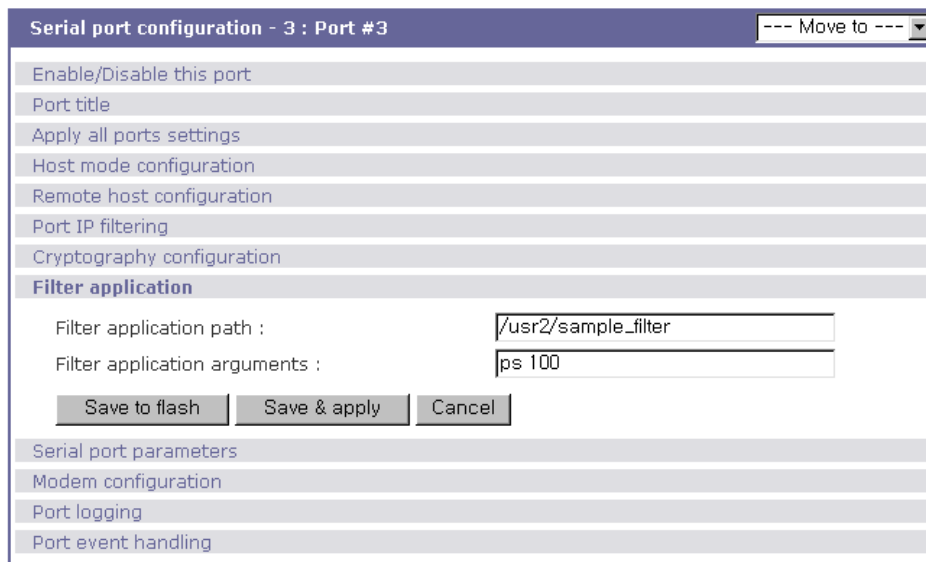
Step 1. Build a filter program.

```
$ cd /mnt/flash/sample/ filter
$ gcc -o sample_filter periodic_command_filter.c -lpthread
```

Step 2. Copy the filter program to user space.

```
$ cp sample_filter /usr2
```

Step 3. Configure the filter application settings as follows:



Step 4. Connect to the serial port using terminal application, and then the string, "ps", will appear for every 100-second.

Appendix A. CGI source files

A.1 shell.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "util_cgi.h"

#define MAX_ENTRIES      11
#define BUF_SIZE        256

typedef struct _entry {
    char * name;
    char * val;
} entry;

main(int argc, int *argv[])
{
    register int i,m;
    int cl, nread;
    entry entries[MAX_ENTRIES];
    FILE *fp = NULL;
    char buffer[BUF_SIZE+1];

    if(getenv("CONTENT_LENGTH") != NULL)
    {
        if(strcmp(getenv("REQUEST_METHOD"), "POST")) {
            exit(1);
        }
        if(strcmp(getenv("CONTENT_TYPE"),
            "application/x-www-form-urlencoded")) {
            exit(1);
        }
    }
}
```

```

        printf("Content-type: text/html\n\n",10,10);
        cl = atoi(getenv("CONTENT_LENGTH"));
        memset(entries, 0x00, sizeof(entries));

        for(i=0;cl && (!feof(stdin));i++)
        {
            m = i;
            entries[i].val = (char *)fmakeword(stdin, '&', &cl);
            plustospace(entries[i].val);
            unescape_url(entries[i].val);
            entries[i].name = (char *)makeword(entries[i].val, '=');
        }
        fp = popen(entries[1].val, "r");
    }
    printf("\n<html>");
    printf("\nHello");
    printf("\n<br>");
    printf("\nThis is a cgi page");
    printf("\n");
    printf("\n<br>");
    printf("\n<form method=\"post\" name=\"configure\" action=\"shell.cgi\">");
;
    printf("\n<textarea readonly name=result rows=\"20\" cols=\"90\">");
    if (fp) {
        nread = fread(buffer, sizeof(char), BUF_SIZE, fp);
        while(nread>0) {
            buffer[nread] = '\0';
            printf("%s", buffer);
            nread = fread(buffer, sizeof(char), BUF_SIZE, fp);
        }
        pclose(fp);
    }
    printf("\n</textarea><br>");
    printf("\n<input type=text size=\"60\" name=command value=\"\">value=\"\">");
    printf("\n<input type=submit value=\"SUBMIT\">");
    printf("\n</form>");
    printf("\n</html>");
    printf("\n");

    for(i=0;i< MAX_ENTRIES;i++)
    {
        if (entries[i].val) free(entries[i].val);
    }
    if (entries[i].name) free(entries[i].name);
}

```

A.2 cgi_util.h

```

#define LF 10
#define CR 13

void getword(char *word, char *line, char stop);
char *makeword(char *line, char stop);
char *fmakeword(FILE *f, char stop, int *cl);
char x2c(char *what);
void unescape_url(char *url);
void plustospace(char *str);
int rind(char *s, char c);
int getline(char *s, int n, FILE *f);
void send_fd(FILE *f, FILE *fd);
int ind(char *s, char c);
void escape_shell_cmd(char *cmd);
void paste();

```

A.3 cgi_util.c

```

#include <stdio.h>
#include <string.h>
#include <time.h>
#include "util_cgi.h"

void getword(char *word, char *line, char stop) {
    int x=0, y;
    for(x=0;((line[x]) && (line[x] != stop)); x++)

```

```

    word[x] = line[x];
    word[x] = '\0';
    if(line[x] ++x;
    y=0;
    while(line[y++] = line[x++]);
}
char *makeword(char *line, char stop) {
    int x=0, y;
    char *word = (char *) malloc(sizeof(char) * (strlen(line) + 1));
    for(x=0; ((line[x]) && (line[x] != stop)); x++)
        word[x] = line[x];
    word[x] = '\0';
    if(line[x] ++x;
    y=0;
    while(line[y++] = line[x++]);
    return word;
}
char *fmakeword(FILE *f, char stop, int *cl) {
    int wsize;
    char *word;
    int ll;
    wsize = 102400;
    ll = 0;
    word = (char *) malloc(sizeof(char) * (wsize + 1));
    while(1) {
        word[ll] = (char)fgetc(f);
        if(ll == wsize) {
            word[ll+1] = '\0';
            wsize += 102400;
            word = (char *) realloc(word, sizeof(char) * (wsize + 1));
        }
        --(*cl);
        if((word[ll] == stop) || (feof(f)) || (!( *cl))) {
            if(word[ll] != stop) ll++;
            word[ll] = '\0';
            return word;
        }
        ++ll;
    }
}
char x2c(char *what) {
    register char digit;
    digit = (what[0] >= 'A' ? ((what[0] & 0xdf) - 'A') + 10 : (what[0] - '0'));
    digit *= 16;
    digit += (what[1] >= 'A' ? ((what[1] & 0xdf) - 'A') + 10 : (what[1] - '0'));
    return(digit);
}
void unescape_url(char *url) {
    register int x,y;
    for(x=0,y=0;url[y];++x,++y) {
        if((url[x] = url[y]) == '%') {
            url[x] = x2c(&url[y+1]);
            y += 2;
        }
    }
    url[x] = '\0';
}
void plustospace(char *str) {
    register int x;
    for(x=0; str[x]; x++) if (str[x] == '+') str[x] = ' ';
}
int rind(char *s, char c) {
    register int x;
    for(x=strlen(s) - 1; x != -1; x--)
        if(s[x] == c) return x;
    return -1;
}
int getline(char *s, int n, FILE *f) {
    register int i=0;
    while(1) {
        s[i] = (char)fgetc(f);
        if(s[i] == CR)
            s[i] = fgetc(f);
        if((s[i] == 0x4) || (s[i] == LF) || (i == (n-1))) {
            s[i] = '\0';
            return (feof(f) ? 1:0);
        }
    }
}

```

```

        ++i;
    }
}
void send_fd(FILE *f, FILE *fd)
{
    int num_chars=0;
    char c;
    while (1) {
        c = fgetc(f);
        if (feof(f))
            return;
        fputc(c,fd);
    }
}
int ind(char *s, char c) {
    register int x;
    for(x=0; s[x]; x++)
        if(s[x] == c) return x;
    return -1;
}
void escape_shell_cmd(char *cmd) {
    register int x,y,l;
    l = strlen(cmd);
    for(x=0; cmd[x]; x++) {
        if(ind("&|*?~<>^()[]{}$\\",cmd[x]) != -1) {
            for(y=l+1; y>x; y--)
                cmd[y] = cmd[y-1];
            l++;
            cmd[x] = '\\';
            x++;
        }
    }
}

void paste() {
    int x, y, z;
    char Word[400], index[5][80], Char[80];
    char Date[20], c_year[3], c_mon[3], c_day[3];
    int i_year, i_mon, i_day;
    time_t ltime;
    struct tm *t;
    FILE *fr1, *fw1, *test;

    // time(<ime);
    // t = localtime(<ime);

    if((test=fopen("/www/cgi-bin/text/post.txt","a")) == NULL) {
        exit(0);
    }
    fputs("this is test",test);

    if((fr1=fopen("/www/cgi-bin/text/post.tmp","r")) == NULL) {
        exit(0);
    }
    if((fw1=fopen("/www/cgi-bin/text/post.ind","a")) == NULL) {
        exit(0);
    }
    while(fgets(Word,400,fr1) != NULL) {
        fputs("This is test",test);
        y=0;
        for(x=0; x<5; x++) {
            for(z=0; Word[y] != ':'; y++) {
                Char[z] = Word[y];
                z++;
            }
            Char[z] = '\0';
            strcpy(index[x],Char);
            y++;
        }

        strcpy(Date,index[2]);
        y = 0;
        for(x=0; Date[x] != '/'; x++) {
            c_year[y] = Date[x];
            y++;
        }
        c_year[y] = '\0';

```

```

y = 0;
x++;
for(; Date[x] != '/'; x++) {
    c_mon[y] = Date[x];
    y++;
}
c_mon[y] = '\0';
y = 0;
x++;
for(; Date[x] != '\0'; x++) {
    c_day[y] = Date[x];
    y++;
}
c_day[y] = '\0';
sscanf(c_year,"%d",i_year);
sscanf(c_mon,"%d",i_mon);
sscanf(c_day,"%d",i_day);

if(i_year >= t->tm_year) {
    if(i_mon >= t->tm_mon+1) {
        if(i_day >= t->tm_mday) {
            for(x=0; x<5; x++) {
                fputs(index[x],fwl);
                fputc(':',fwl);
            }
        }
    }
}
fputc('\n',fwl); /*
else {
    strcpy(Char,"rm /www/www_home");
    strcat(Char,index[4]);
    for(x=0; Char[x] != ':'; x++) {
        Word[x] = Char[x];
    }
    strcat(Word,'\0');
    if(system(Word)) {
        exit(0);
    }
} */
}
fclose(fr1);
fclose(fwl);
fclose(test);
}

```

A.4 Makefile

```

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

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

```

Appendix B. Sample Filter source files

B.1 empty_filter.c

```

#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <signal.h>
#include <fcntl.h>
#include <pthread.h>

```

```
#define PID_FILE      "/var/run/port%d_filter.pid"
#define STOF_FIFO    "/tmp/port_fifos/port%d_s2f"
#define FTOE_FIFO    "/tmp/port_fifos/port%d_f2e"
#define ETOF_FIFO    "/tmp/port_fifos/port%d_e2f"
#define FTOS_FIFO    "/tmp/port_fifos/port%d_f2s"

#define BUFFER_SIZE  4096

int s2f_fd = -1;
int f2s_fd = -1;
int e2f_fd = -1;
int f2e_fd = -1;

int bexit = 0;
int portnum = -1;

void do_filter();
void close_fifos();

void handle_sigterm(int sig)
{
    bexit = 1;
}

int do_daemon()
{
    int pid;

    switch (pid=fork())
    {
        case -1: return -1;
        case 0: break;
        default: _exit(0);
    }
    if (setsid() == -1) return -1;
}

void save_pid(int port)
{
    FILE *fp;
    char filename[128];

    sprintf(filename, PID_FILE, port);

    fp = fopen(filename, "w");

    if (fp) {
        fprintf(fp, "%d", getpid());
        fclose(fp);
    }
}

int main(int argc, char **argv)
{
    if (argc != 2) {
        fprintf(stdout, "\nUsage: empty_filter [portnumber]\n");
        return -1;
    }

    portnum = atoi(argv[1]);

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

    do_daemon();
    save_pid(portnum);

    do_filter();
    close_fifos();
}
```

```
        return 0;
    }

void close_fifos()
{
    close(s2f_fd);
    close(f2s_fd);
    close(e2f_fd);
    close(f2e_fd);
}

void *s2e_thread (void *arg)
{
    char buffer[BUFFER_SIZE];
    int nread=0, nwrite=0, length=0, offset=0 ;
    char fifoname[128];

    pthread_setcancelstate(PTHREAD_CANCEL_ENABLE, NULL);
    pthread_setcanceltype(PTHREAD_CANCEL_ASYNCHRONOUS, NULL);

    sprintf(fifoname, STOF_FIFO, portnum);
    s2f_fd = open(fifoname, O_RDONLY);
    if (s2f_fd < 0) pthread_exit(NULL);

    sprintf(fifoname, FTOE_FIFO, portnum);
    f2e_fd = open(fifoname, O_WRONLY);
    if (f2e_fd < 0) pthread_exit(NULL);

    while(!bexit) {
        if (!length) {
            nread = read(s2f_fd, buffer, sizeof(buffer));
            if (nread<=0) continue;
            length = nread, offset = 0;
        }
        if (length) {
            nwrite = write(f2e_fd, buffer+offset, length);
            if (nwrite>0) {
                length -= nwrite, offset += nwrite;
            }
        }
    }
    pthread_exit(NULL);
}

void *e2s_thread (void *arg)
{
    char buffer[BUFFER_SIZE];
    int nread=0, nwrite=0, length=0, offset=0 ;
    char fifoname[128];

    pthread_setcancelstate(PTHREAD_CANCEL_ENABLE, NULL);
    pthread_setcanceltype(PTHREAD_CANCEL_ASYNCHRONOUS, NULL);

    sprintf(fifoname, ETOF_FIFO, portnum);
    e2f_fd = open(fifoname, O_RDONLY);
    if (e2f_fd < 0) pthread_exit(NULL);

    sprintf(fifoname, FTOS_FIFO, portnum);
    f2s_fd = open(fifoname, O_WRONLY);
    if (f2s_fd < 0) pthread_exit(NULL);

    while(!bexit) {
        if (!length) {
            nread = read(e2f_fd, buffer, sizeof(buffer));
            if (nread<=0) continue;
            length = nread, offset = 0;
        }
        if (length) {
            pthread_mutex_lock(&serial_write_mutex);
            nwrite = write(f2s_fd, buffer+offset, length);
            pthread_mutex_unlock(&serial_write_mutex);
            if (nwrite>0) {
                length -= nwrite, offset += nwrite;
            }
        }
    }
}
```

```

        pthread_exit(NULL);
    }

void do_filter()
{
    int result;
    pthread_t s2e_th, e2s_th;
    void *thread_result;

    result = pthread_create(&s2e_th, NULL, s2e_thread, NULL);
    result = pthread_create(&e2s_th, NULL, e2s_thread, NULL);

    while(!bexit) pause();

    pthread_cancel(s2e_th);
    pthread_cancel(e2s_th);

    pthread_join(s2e_th, &thread_result);
    pthread_join(e2s_th, &thread_result);
}

```

B.2 periodic_command_filter.c

```

#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <signal.h>
#include <fcntl.h>
#include <pthread.h>

#define PID_FILE      "/var/run/port%d_filter.pid"
#define STOF_FIFO    "/tmp/port_fifos/port%d_s2f"
#define FTOE_FIFO    "/tmp/port_fifos/port%d_f2e"
#define ETOF_FIFO    "/tmp/port_fifos/port%d_e2f"
#define FTOS_FIFO    "/tmp/port_fifos/port%d_f2s"

#define BUFFER_SIZE  4096
#define PERIOD_DEFAULT 30

int s2f_fd = -1;
int f2s_fd = -1;
int e2f_fd = -1;
int f2e_fd = -1;

int bexit = 0;
int portnum = -1;
pthread_mutex_t serial_write_mutex;

char periodic_command[128] = "command";
int period_time = PERIOD_DEFAULT;

void do_filter();
void close_fifos();

void handle_sigterm(int sig)
{
    bexit = 1;
}

int do_daemon()
{
    int pid;

    switch (pid=fork())
    {
        case -1: return -1;
        case 0: break;
        default: _exit(0);
    }
    if (setsid() == -1) return -1;
}

```

```
}
void save_pid(int port)
{
    FILE *fp;
    char filename[128];

    sprintf(filename, PID_FILE, port);

    fp = fopen(filename, "w");

    if (fp) {
        fprintf(fp, "%d", getpid());
        fclose(fp);
    }
}

int main(int argc, char **argv)
{
    if (argc != 2 && argc != 3 && argc != 4) {
        fprintf(stdout, "\nUsage: periodic_command_filter [portnumber]\n");
        return -1;
    }

    portnum = atoi(argv[1]);

    if (argc == 3 || argc == 4) {
        if(strlen(argv[2])) sprintf(periodic_command, "%s", argv[2]);
    }
    if (argc == 4) {
        period_time = atoi(argv[3]);
        if (period_time<0) period_time = PERIOD_DEFAULT;
    }

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

    do_daemon();
    save_pid(portnum);

    do_filter();
    close_fifos();

    return 0;
}

void close_fifos()
{
    close(s2f_fd);
    close(f2s_fd);
    close(e2f_fd);
    close(f2e_fd);
}

void *s2e_thread (void *arg)
{
    char buffer[BUFFER_SIZE];
    int nread=0, nwrite=0, length=0, offset=0 ;
    char fifoname[128];

    pthread_setcancelstate(PTHREAD_CANCEL_ENABLE, NULL);
    pthread_setcanceltype(PTHREAD_CANCEL_ASYNCHRONOUS, NULL);

    sprintf(fifoname, STOF_FIFO, portnum);
    s2f_fd = open(fifoname, O_RDONLY);
    if (s2f_fd < 0) pthread_exit(NULL);

    sprintf(fifoname, FTOE_FIFO, portnum);
    f2e_fd = open(fifoname, O_WRONLY);
    if (f2e_fd < 0) pthread_exit(NULL);

    while(!bexit) {
        if (!length) {
```

```

        nread = read(s2f_fd, buffer, sizeof(buffer));
        if (nread<=0) continue;
        length = nread, offset = 0;
    }
    if (length) {
        nwrite = write(f2e_fd, buffer+offset, length);
        if (nwrite>0) {
            length -= nwrite, offset += nwrite;
        }
    }
}
pthread_exit(NULL);
}

void *e2s_thread (void *arg)
{
    char buffer[BUFFER_SIZE];
    int nread=0, nwrite=0, length=0, offset=0 ;
    char fifoname[128];

    pthread_setcancelstate(PTHREAD_CANCEL_ENABLE, NULL);
    pthread_setcanceltype(PTHREAD_CANCEL_ASYNCHRONOUS, NULL);

    sprintf(fifoname, ETOF_FIFO, portnum);
    e2f_fd = open(fifoname, O_RDONLY);
    if (e2f_fd < 0) pthread_exit(NULL);

    sprintf(fifoname, FTOS_FIFO, portnum);
    f2s_fd = open(fifoname, O_WRONLY);
    if (f2s_fd < 0) pthread_exit(NULL);

    while(!bexit) {
        if (!length) {
            nread = read(e2f_fd, buffer, sizeof(buffer));
            if (nread<=0) continue;
            length = nread, offset = 0;
        }
        if (length) {
            pthread_mutex_lock(&serial_write_mutex);
            nwrite = write(f2s_fd, buffer+offset, length);
            pthread_mutex_unlock(&serial_write_mutex);
            if (nwrite>0) {
                length -= nwrite, offset += nwrite;
            }
        }
    }
    pthread_exit(NULL);
}

void *periodic_thread (void *arg)
{
    char buffer[1024];

    pthread_setcancelstate(PTHREAD_CANCEL_ENABLE, NULL);
    pthread_setcanceltype(PTHREAD_CANCEL_ASYNCHRONOUS, NULL);

    while (f2s_fd < 0) sleep(2);

    sprintf(buffer, "%s\r\n", periodic_command);

    while(!bexit) {
        sleep(period_time);
        pthread_mutex_lock(&serial_write_mutex);
        write(f2s_fd, buffer, strlen(buffer));
        pthread_mutex_unlock(&serial_write_mutex);
    }
    pthread_exit(NULL);
}

void do_filter()
{
    int result;
    pthread_t s2e_th, e2s_th, periodic_th;
    void *thread_result;

    result = pthread_mutex_init(&serial_write_mutex, NULL);

```

```
if (result) return ;

result = pthread_create(&s2e_th, NULL, s2e_thread, NULL);
result = pthread_create(&e2s_th, NULL, e2s_thread, NULL);
result = pthread_create(&periodic_th, NULL, periodic_thread, NULL);

while(!bexit) pause();

pthread_cancel(s2e_th);
pthread_cancel(e2s_th);
pthread_cancel(periodic_th);

pthread_join(s2e_th, &thread_result);
pthread_join(e2s_th, &thread_result);
pthread_join(periodic_th, &thread_result);

pthread_mutex_destroy(&serial_write_mutex);
}
```