SC communication program manual

Shige Matsuno
University of Hawaii, Manoa

9 May 1994

DIR-8-94

1 Introduction

In this document, details of the DUMAND string controller (SC) communication program will be discussed. This document mainly concerns the internal structure of the program, namely how it is organized, how it works, etc. On the usage of this program, one should consult with a memo written by Dennis[1].

This program is meant to control and to monitor all the aspects of the SC. Also, it is meant to mediate the communication between the modules and the operator and between the SC environmental module (SCEM) and the operator. It is designed to be able to communicate with all the modules in the string simultaneously and relay the response from them to the shore operator. At the same time, it can communicate with the SCEM, communicate and control the SC digitizer, and monitor the SC parameters like temperature, electric current, and voltage.

The program down-loading scheme is included in this program so that we can use the different version of the software for the SC, the SCEM, or the modules if necessary. "Kermit" communication protocol is used as the standard file transfer protocol.

The shore operator has to use predetermined commands with a command confirmation scheme to communicate this program. These commands are described in detail in Dennis's memo[1]. The commands are rather cryptic, but meant to be used through an interface program which runs on the shore control computer and supposes to be more user firendly. The modules communicate to the SC using the similar but different set of commands, which is descried in another memo[2]. The details on the hardware in the SC can be found in the other manual[3][4][5].

This program is written for OS-9 operating system primary using C language. Some I/O related parts like device driver and the system interface part of the program are written in Assembler language, to have a better control over the hardware. It is quite possible to use C in the entire software, but it is decided not to do this because of the lack of sample programs for I/O and system interface routines.

2 OS-9

OS-9 is an UNIX like operating system for the small computer system[6]. As in the case of UNIX, it is a multi-user multi-task operating system. But unlike UNIX, this is a real-time operating system, namely you can use hardware interrupts in the system, though the interrupt handling routine is usually hidden inside the device driver and not directly accessible from users.

This operating system has been selected to be an universal OS for the entire underwater software of the DUMAND project. We wanted to use a single operating system for all the underwater software to avoid their maintenance nightmare. The OS-9 system has been chosen because it is a real-time operating system, supports higher level languages like C and FORTRAN, and is popular among the Motorola's 68k based computer, which was chosen to be used as a control computer for the modules and the SC.

Many features of this OS have been utilized in the communication program to make the program development time shorter, to make the resulting program's memory requirement smaller, and to go around the certain system limitation. One example of this is the extensive use of the pipe for the communication between tasks, which consist the actual program.

3 program structure

The communication software consists of three independent tasks. The first one is "comm", the second one is "comm_om", and the third one is "sctimer". Also there is another task called "comm_kill" which is designed to terminate "comm" without using its standard i/o port and usually dormant. These tasks communicate each other using pipes and events, which are some of the inter-task communication schemes supported by the OS-9. The details on these tasks are descried below.

3.1 comm

This is the main part of the communication software. It communicates with the shore operator through the slow laser communication line and handles all the incoming commands and the replies to the operator. It also spawns other tasks necessary to communicate with the modules and to initiate time execution routine. This task handles i/o's to the ADC, DAC, and interface to the digitizer and monitors the SC's internal parameter.

All the commands from the shore operator is received, interpreted, and executed by this task. When it receives a command, it echos back the command and waits for a confirmation from the operator ("OK") before actually executing the command. If the module i/o is necessary to execute the command, module i/o commands are sent to "comm_om" task through the pipe. Reply from all "comm_om" task initiated by this task is routed through unnamed pipe, received, and relayed to the shore operator by "comm".

Upon receiving the termination command ($KL) from the shore operator, "comm" terminates all the tasks it initiated, closes all the paths opened, and terminates itself. To kill the forked task "comm_om", "comm" will send a character string, "END", to it. There is another way to kill "comm", which is using "comm_kill" program. "comm" sets up an event communication scheme upon its initiation and keeps checking whether the event flag has been set or not whenever it finishes one command execution or an execution of timer initiated loop. What the "comm_kill" program does is set the appropriate event flag and let "comm" kill itself.

The source list of this "comm" consists of three separate files, comm_inc.c, comm_func.c, and comm.c. The data structures and functions used in this task is listed in section 6.1 and section 6.2, respectively.

comm_inc.c ; This file contains global variables, definitions, constants, and functions used throughout the "comm" program. This file also contains I/O intercept function, which generates software interrupt whenever "comm" receives a character either from one of i/o paths or from the pipes.

comm_func.c ; This file contains many functions which will be mainly called from the main routine or one of the functions in "comm.c". These function actually execute the individual command after they are interpreted by the routines in "comm.c".

comm.c ; This file contains main routine, initial condition setting function, main command interpreter function, etc. All the command hand shake scheme between the task and the shore operator and main command assignment routine is included in this file.

3.2 comm_om

This is the task which handles the communication to the modules, through the i/o device driver. This task is written to deal with four modules which share one module communication/power board. This task communicates with the main task, "comm", through the pipes. It will receive command to be sent to the modules through the named pipe which is specifically assigned to the individual task and send the module reply back to "comm" through the unnamed pipe, which is common to all the tasks.

All the command hand shaking scheme between the SC and the modules will be dealt in this task along with the check for improper reply from the modules. For example, if the module echo back of a command is not correct, this task will send a special message to "comm" so that it can send an appropriate error message to the shore operator. Also if the module did not reply within a certain time, I/O time-out signal will be sent back.

ROM'ed version has only one "comm_om" in it and "comm" initiate the same program seven times for different module i/o cards. But, this does not mean the same task image file will be copied to the RAM seven times. OS-9 is clever enough to share the same image file in ROM among the tasks with separate data buffers for them set up in the RAM.

The source list of this task is contained in a file "comm_om.c".

3.3 sctimer

This is the timer routine for the "comm" routine. "comm" spawns this routine when its standard input port is rerouted to unnamed pipe. Whenever this routine outputs something (actually this is done every second and the output is fixed), they are routed to "comm" through the pipe. Upon receiving this output of the "sctimer", "comm" generates a software interrupt and initiate an execution of timer part of its routine, like processing "D" commands and the error condition checking.

3.4 comm_kill

This is a task which kills "comm" program. It is meant to use through the ship-board modem communication channel to stop the existing "comm" program. This is necessary because "comm" will starts up on the main communication port (/term) whenever the SC power is turned on. To start the program from the different port from /term, one has to stop the program currently running, first.

4 program feature

4.1 auto program start up

The "comm" program is required to start up automatically upon the SC power up without any action from the shore operator. Because this is the program we will run in the SC anyway and we want to make it run even if the communication between the SC and the operator does not work properly at first. To accomplish this, one has to go into the detail of the operating system and use the way the system starts up its own shell.

When the system boots up, OS-9 executes a task named "sysgo" after it finishes all the necessary system initiation. What this "sysgo" usually does is to keep forking the OS-9 shell so that the shell is alway running even if one execution of it had been terminated. In the ROM'ed version of the communication software, this "sysgo" is modified such a way that it will start "comm" program, before going into the shell forking loop. This special version of "sysgo" is in the file named "sysgo_scc".

4.2 data modules

The "comm" uses a few data module to get a list of initial set up commands, get default values for the modules, and get a list of commands to be executed when there is no command received from the operator within a certain period of time.

Originally, these information were intended to be stored in plain ASCII files. But in the process of the ROM'nize the program, it was found that there is no easy way to put the ASCII files in ROM and make the system recognize them. The former can be done by just merging the ASCII files with the program, but the later can not. This is the reason why the special data modules have to be used to let the program receive the information necessary. The structure of this data module is presented in section 6.1

initlist ; This is the list of the commands which should be executed upon the start up of the "comm" program. It includes commands to power off all the modules, to set up LED receiver threshold to a certain value, and to initialize the SC digitizer.

The command which requires to communicate to the module should not be included in this list. The reason is that the modules are not turned their power on at "comm" start up by default and the modules require some time before they can properly receive a command even if the module turning on commands are included in the list.

deflist ; This is the list of the module default values, including the type of the module which affects the commands "comm_om" can send to the module. For example, one can not send a CM command unless the module being communicated is listed as a CM in this list.

autoexec ; This is the list of commands to be executed when there is no command from the shore operator in predetermined time. This usually include the command to power all the modules and to set the LED threshold to the proper value. This is another safe guard scheme for the communication failure. The same kind of arrengement had been implemented into the module control software[2].

5 other detail

5.1 related program

We had to develop a few other programs to make the program explained above to be usable. One of them are the "sysgo" described above and others are the device driver necessary to use i/o port. We have developed module communication and power supply board[5] which uses a UART chip the OS-9 we had did not support. Also, the communication return path to the shore operator is using the fast laser data structure, which requires the characters to be output to the digitizer board in parallel along with usual serial one for the testing. So, we have to modify the existing device driver for the serial port to be able to handle this.

sc8x50 ; This is the device driver for the Intel's 8350/8550 UART which is used on the CPU board as a serial interface chip. This driver has been modified to output the character to the Boston's digitizer board in parallel.

uart ; This is the device driver developed for handling the i/o to the modules. It is written for an UART chip 81C17 made by the Standard Microsystems, with a full support of its interrupt scheme, baud rate, stop bits, parity, bits per word setting, etc. It can handle the communication speed from 300 to 9600 baud. But it does not support the baud rate change while the port is in use, namely one can not change the baud rate once the port has been opened with the current device driver. One has to change the device descriptor to accomplish this.

In terms of ensuring the continuous communication, module i/o ports usually disable XOn/XOff communication hand shaking. If they are enabled and the noise faked the XOn signal, the i/o port will hang up because it will wait for XOff signal before the normal communication can be resumed.

Because we are using only one interrupt line for all the module i/o ports, it is conceivable that the interrupt from the multiple module overlapped each other. The system's interrupt handling is initiated by a rising edge of the interrupt signal line, so the overlapped interrupt will be ignored and there will be no more interrupt handling thereafter because the interrupt line is always on. The driver had been incoorporated a logic to force a local interrupt signal line to be low every the driver's interrupt handling routine has been executed, thus avoiding the unprocessed interrupt problem.

6 structures and functions

6.1 data structures

These are the data structures used in "comm" program.

```
/* data of the tasks for communicating to the modules */
 struct tasks {  short int   pid ;           process id of the task
                 short int   p_no ;          path number of the pipe
                 char        p_name[12] ;    pipe name used to communicate to the task
                 char        *io_name[4] }   i/o names for this task

/* data for the modules in the string */
 struct modules {  enum m_type     type ;                      module type
                   unsigned short  hv_def, t1_def, t2_def ;    default values
                   unsigned short  hv_act, t1_act, t2_act ;    actual value
                   unsigned short  error, t_out, others ;      error counters
                   unsigned short  pd_thr, status }            LED threshold and

/* "D" and "A" command data structure */
 struct d_str {  char            comm[4] ;        command buffer
                 unsigned long   mmb ;            < mmb > buffer
                 int             dt,trep ;        time interval
                 time_t          tlastm, tlrep ;  last time
                 float           sumn[32] ;       data buffer for "A" command
                 float           sumx[32] ;       data buffer for "A" command
                 float           sumxx[32] }      data buffer for "A" command

/* structure for the data modules used in "comm" */
 struct data_mod {  struct modhcom   _mh ;          data module header
                    long             md_offset ;    offset of the data
                    char             data }         data buffer
```

6.2 functions

Following is the list of functions used in "comm" program.

- functions in comm_inc.c

type & name	function
void wipe(char *)	wipe character array with '\0'
int comp_str(char *,char *)	string comparison function
void sighand(register int)	signal handling (software interrupt)

- functions in comm_func.c

type & name	function
void strip_mmb(int,int,int,int,int)	strip module modifier bits $< mmb >$ from a command
void recover(int)	purge all the characters left in a port
int c_status(int)	report communication status
void p_m_line(int, struct modules *)	print out current module setting
int c_default(int)	module default display/edit
int ADC_read(int,int)	read ADC on module comm/power board
int BU_contrl(char,int,long int)	BU digitizer control interface
int set_input(long int)	set module input enable switch
int set_thr(long int,long int)	set LED receiver threshold
int read_spy(int *,int *,short int)	read spy port (BU digitizer)
int read_phase(short,struct d_str *)	read phase ADC
void auto_thr(long int)	set receiver threshold automatically
void a_thr_2(long int)	auto_threhold #2
int l_comm(int)	LED receiver control
int dig_cont(int)	digitizer control
int chk_hb()	check clock heart beat
int chk_sync()	check sync error bit of the digitizer
int read_temp(short,struct d_str *)	read SC temperature
int read_para(short,struct d_str *,short)	read parallel port
int read_OM(int,short)	OM "R" command handling
int CM_comm(int)	CM command handling
int OM_comm(int)	OM command handling
int d_comm(int,int)	"D" and "A"command handling function
int p_through(int,int)	communication pass through mode
int default_comm(int)	default $< mmb >$ handling
int power(int,short)	module power control
int on_power(int)	module power on
int em_ack(int)	emergency acknowledgement handling
int test_oc(int,int)	test module over current
int down_load(int,int)	program download to module
int d_l_em(int,int)	program download to SCEM
int suspend(int)	"comm" suspend command
int s_kermit(int,int)	program download from the shore
int d_chirp(int)	chirp file download to SCEM

- functions in comm.c

type & name	function
void scc_init()	initialize "comm" program
int sccquit()	terminate "comm" program
int comm_acc()	shore communication hand shake
void comm_exe(int)	shore command execution
void mod_reply(int)	module reply handling

void EM_reply()	SCEM reply handling
void prexit()	print help information and quit

References

[1] D. Nicklaus. *"DUMADND Host Computer Status Monitor Software External Interface Specifications"*

[2] S. Matsuno. *"Communication protocol between the SC and the modules"*

[3] E. Hazen et. al. *"DUMAND string Controller Digitizer System Design Specification"* DIR-1-92

[4] D. Orlov *"DUMAND Clock Signal Generator"*

[5] M. Mignard *"SC hardware manual"*

[6] Microware Systems Co. *"OS-9 Technical Manual"* Oct. 1989