A Logging Design Based on Embedded Software

Hai Luo(✉), Yikun Wu, Long Ma, and Ting Zheng

The Twentieth Research Institute of China Electronics Technology Group Corporation, Xi'an, China
kaito1992@126.com

Abstract. The black-box feature of embedded software is always a problem that software developers need to face in the development and testing process, and it is often impossible to locate and reproduce the problem in time when the software fails to run. In this paper, based on the characteristics of embedded software, a logging function is designed to record the debugging information of the software at all times during the operation of the software, so that the development and testing personnel can analyze and monitor the operation status of the software.

Keywords: logging · debugging information · Real-time

1 Introduction

Embedded software is a kind of software running in the embedded system, which belongs to a kind of computer software. Embedded system is generally composed of four parts: microprocessor, hardware device, embedded operating system and user's application, etc. It emphasizes high real-time, reliability and stability of program operation [1].

Embedded software can usually be divided into three main categories, which are system software, application software and support software. System software, such as embedded middleware, device drivers, etc., provides interface support for embedded applications and controls and manages embedded system resources [2]. Application software, such as flight control software, electronic map software, etc., are generally oriented to specific application areas, define the main functions and uses of embedded devices, responsible for interacting with users, and belong to the upper layer software in the embedded system [3]. Support software represents the tool software to assist development, such as online simulation tools, system analysis and design tools, etc. [4].

Embedded software as a type of software with relatively high requirements for real-time, its operation in the process of business processing response is more demanding, especially when it comes to safety-related areas such as aerospace, automotive electronics, etc. [5]. And its operation process is not like the traditional desktop software can be real-time monitoring of memory, operating status and other information, embedded software for the user is equivalent to a "black box", only care about the effectiveness of the output, the process of generating the output is not concerned, so the operating state of embedded software for the user is an uncontrollable state [6].

https://doi.org/10.2991/978-94-6463-198-2_158

In this paper, we design a logging mechanism to detect the various states of embedded software during operation for users to read and grasp the "black box" characteristic of embedded software.

2 Introduction to Embedded Software Debugging Information

In the process of embedded software running, usually through an external serial port connected to the computer screen, through the computer screen can be observed when the embedded software running debugging information [7]. The debugging information is displayed in the process of running the program involving "printf" and "logMsg" key print, these information can be used in the coding stage to debug some key position tips for developers to test. Developers can add their own concerns to the debugging information according to their actual debugging situation. For example [8].

0x84fc4000(tReadSWMsg):RecvSWMsgType:0-1-38, the above information is a common kind of debug information output from serial port in embedded OS VxWorks, the first "0x84fc4000" means the debug information corresponds to "tReadSWMsg" indicates the process name corresponding to the current debugging information, which can be freely defined by the user during the development process; "RecvSWMsgType:0-1-38" indicates the content The content of "RecvSWMsgType:0-1-38" is completely defined by the user through logMsg function, which can output any string information or variable information [9]. The print information is a common online debugging tool for embedded software, which can add relevant print information in any part of the code, and can play the role of real-time monitoring the running status of the software in the case of external serial port [10].

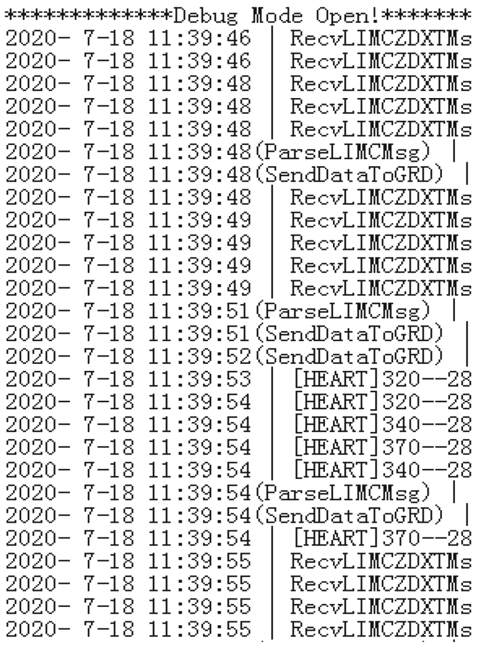
The defect of this debugging method is that you must artificially observe the serial port print information to understand the operating status of the embedded software, in the absence of an external screen and debugging personnel, and back to the black box properties of embedded software [11].

3 Design of Logging Function Based on Debugging Information

The logging function designed in this paper is a logging method designed for the embedded software in the normal operation process, the outside world can not monitor the software running status, in order to make the embedded software in the running process to record the key debugging information for subsequent extraction. The design principle is to ensure that the operating system is equipped with enough storage space to store the log, and the design is inserted in the code, and the relevant program information is written to the log when running to some key branches. The logical design of the logging program is as follows.

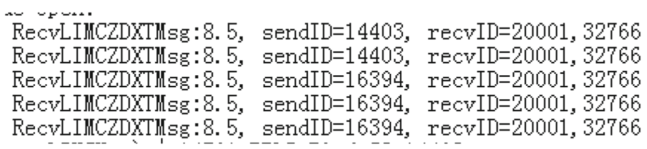
We will write the log function named WriteLogFile, the function has only one formal parameter we will name it cLogStr, which means the content of the written log string, its type is unsigned char. We use "WriteLogFile(cLogStr)" to call this function to achieve the function of writing logs, when calling this function, you need to get the current system time through the system function, we take Vxworks5.5 as an example, its function to get the system time is localtime. After getting the system time through this function, use the fopen function to open the log file, and then write the system time and log information together to the log file through the fprinft function, the format of the log content is "year-month-day hour-minute-second | log content", the displayed effect is as follows Fig. 1, Fig. 2 an Fig. 3 shown.

In the process of implementing logging, there is a wide variety of log contents, and we define the format of logs in a macro level, i.e., so log events are recorded according to our pre-agreed format to be accessed by readers later. The specific design of the log content format we use the following Table 1 to achieve.



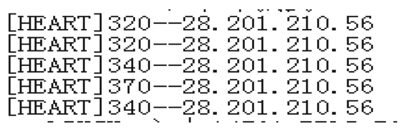
```
*************Debug Mode Open!*******
2020- 7-18 11:39:46 | RecvLIMCZDXTMs
2020- 7-18 11:39:46 | RecvLIMCZDXTMs
2020- 7-18 11:39:48 | RecvLIMCZDXTMs
2020- 7-18 11:39:48 | RecvLIMCZDXTMs
2020- 7-18 11:39:48 | RecvLIMCZDXTMs
2020- 7-18 11:39:48(ParseLIMCMsg) |
2020- 7-18 11:39:48(SendDataToGRD) |
2020- 7-18 11:39:48 | RecvLIMCZDXTMs
2020- 7-18 11:39:49 | RecvLIMCZDXTMs
2020- 7-18 11:39:49 | RecvLIMCZDXTMs
2020- 7-18 11:39:49 | RecvLIMCZDXTMs
2020- 7-18 11:39:49 | RecvLIMCZDXTMs
2020- 7-18 11:39:51(ParseLIMCMsg) |
2020- 7-18 11:39:51(SendDataToGRD) |
2020- 7-18 11:39:52(SendDataToGRD) |
2020- 7-18 11:39:53 | [HEART]320--28
2020- 7-18 11:39:54 | [HEART]320--28
2020- 7-18 11:39:54 | [HEART]340--28
2020- 7-18 11:39:54 | [HEART]370--28
2020- 7-18 11:39:54 | [HEART]340--28
2020- 7-18 11:39:54(ParseLIMCMsg) |
2020- 7-18 11:39:54(SendDataToGRD) |
2020- 7-18 11:39:54 | [HEART]370--28
2020- 7-18 11:39:55 | RecvLIMCZDXTMs
2020- 7-18 11:39:55 | RecvLIMCZDXTMs
2020- 7-18 11:39:55 | RecvLIMCZDXTMs
2020- 7-18 11:39:55 | RecvLIMCZDXTMs
```

Fig. 1. Example diagram of a log file (1)



```
RecvLIMCZDXTMsg:8.5, sendID=14403, recvID=20001,32766
RecvLIMCZDXTMsg:8.5, sendID=14403, recvID=20001,32766
RecvLIMCZDXTMsg:8.5, sendID=16394, recvID=20001,32766
RecvLIMCZDXTMsg:8.5, sendID=16394, recvID=20001,32766
RecvLIMCZDXTMsg:8.5, sendID=16394, recvID=20001,32766
```

Fig. 2. Example diagram of a log file (2)



```
[HEART]320--28.201.210.56
[HEART]320--28.201.210.56
[HEART]340--28.201.210.56
[HEART]370--28.201.210.56
[HEART]340--28.201.210.56
```

Fig. 3. Example diagram of a log file (3)

Table 1. Logging format schematic table

No.	elements	
	Time (Year-Month-Day Hour-Min-Sec)	Event
1	2020-7-18 11:40:39	RecvLIMCZDXTMsg:8.5 ,sendID=14403,recvID =20001,32766
2	2020-7-18 11:42:28	[HEART]320—28.201.210.56
3	2020-7-18 11:43:40	(ParseLIMCMsg)13020 Find C2
4	2020-7-18 11:45:03	(SendDataToGRD) Msg b6 Send Error, Discard
5	……	……

In the design method implementation, we take code insertion, that is, we can call this logging function in the code segment that needs logging, and cut the code segment can be called according to the logging switch or not. The code logic is designed as follows.

First determine whether the switch of logging is on, if it is on, then enter the corresponding branch and store the log content into the corresponding string through the spintf function, e.g.

sprintf(DebugLog, "%s:FileName %s Transfer interrupted!", __FUNCTION__, cFileName);

In this example, "DebugLog" is a string array that stores the log information; "__FUNCTION__" represents the name of the function that executed this statement, so you can locate the exact location when viewing the log.

After confirming that the logging function is to be performed, the program enters a series of steps before writing the log file, first using the fopen function to open the log file and create a file pointer. Finally, use the fprintf function to write all the contents to the file pointer together, and close the file pointer after writing through the fclose function to avoid creating a wild pointer. The final write statement is

fprintf(pFile, "%04d-%02d-%02d %02d:%02d:%02d | %s\r\n", ttime.tm_year+1900, ttime.tm_mon+1, ttime.tm_mday, ttime.tm_hour, ttime_tm_min, ttime_tm_sec, cLogStr);

where pFile is the pointer to the open log file, ttime is the system time structure, and cLogStr is the log message. The specific definition of the ttime structure is shown below.

```
Struct ttime
{
        int tm_sec;     /*second after the minute – [0,59]*/
        int tm_min;     /*minutes after the hour  – [0,59]*/
        int tm_hour;    /*hours after midnight    – [0,23]*/
        int tm_mday;    /*day after the month     – [1,31]*/
        int tm_mon;     /*months since January    – [0,11]*/
        int tm_year;    /*years since 1900*/
        int tm_wday;    /*days since Sunday       – [0,6]*/
        int tm_yday;    /*days since January 1– [0,365]*/
        int tm_isdst;   /*Daylight Saving Time flag*/
}
```

In this case, if the time we need to write is January 20, 2023, the value in the ttime structure that we get through the system function is

ttime_tm.year = 123;
ttime_tm.mon = 0;
ttime_tm.mday = 20;

A conversion is required to get the current specific date.

In practice, the module code function will only be accessed by the program when the logging function is turned on, requiring the designer to place the function code at the point where the logs will appear when designing, and the logging file function can be operated by means of window instructions or configuration files, which also facilitates the use and maintenance of the operator.

The flow of the logging function is as follows Fig. 4 shown.

1) When the program starts to call the logging function, it first checks whether the current logging file is valid, and if it is not, it creates a new logging file and opens the logging file to prepare for writing.
2) Before writing the log, obtain the current system time through the system function, ensure that the time obtained is valid, and write it to the log record file together with the log information.
3) Always detect the switch status of the logging function, when the logging function remains open, the logging file can be opened to record logs at any time, and when the logging function is detected to be closed, the logging file can be closed at the same time, and the program will stop calling related functions synchronously

In addition, frequent logging will cause the log file to become bigger and bigger, so while designing the logging function, this paper also designs the storage mode of the log file to ensure the integrity of the whole function and avoid other defects caused by the introduction of the logging function.

In the design, the size of a single logging file is fixed to 8M, and the maximum number of log files is set to 100. The processing flow is shown in the following Fig. 5.

1) During the execution of the logging function of the program, the size of the log file needs to be checked at any time, and if the size exceeds a certain threshold (in the actual design we recommend using the upper limit of 8M, the file is a file for users

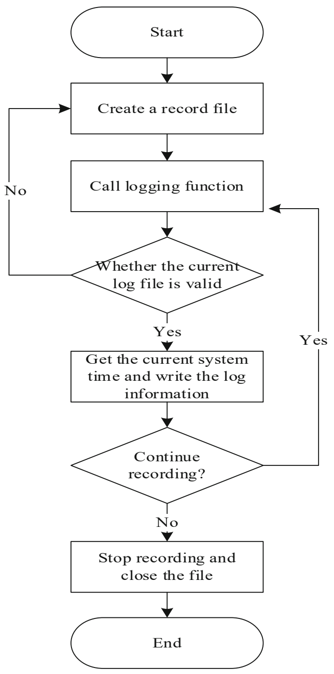

Start
Create a record file
Call logging function
Whether the current log file is valid
No
Yes
Get the current system time and write the log information
Continue recording?
Yes
No
Stop recording and close the file
End

Fig. 4. Flow chart of logging function

to read, for text-based files 8M can store enough entries, and the appropriate size can ensure that the file does not take up too much time and resources when it is extracted that is, opened), a new logging file needs to be recreated.

2) Before creating a new log file, the log file system is scanned and the number of all current log files is counted. If the number of log files exceeds a certain threshold (the recommended limit here is no more than 100, for vxworks systems, too many files will add extra burden to the system and will take up a lot of memory space when the system is detected, affecting the normal operation of the system), the oldest log file is deleted.
3) Create a new logging file and perform logging related functions on the basis of the new file.

In the process of designing the logging function, we not only designed the specific logging from the function itself, but also made certain restrictions on the size of the logging file, the number of logging files and other indicators that affect the system.

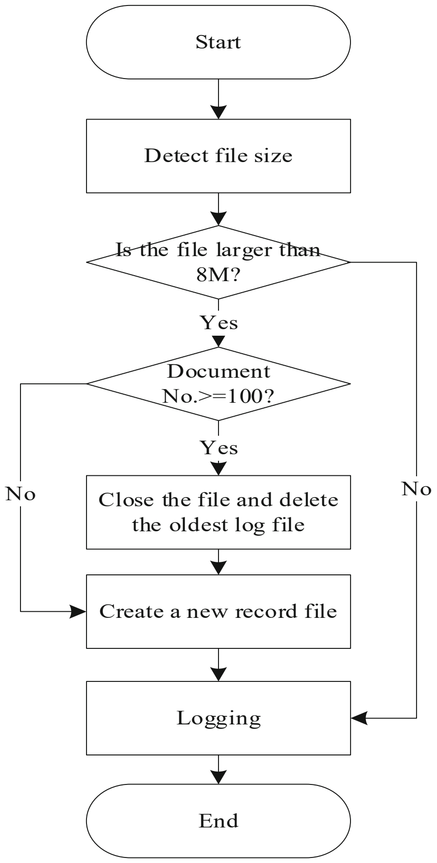


Fig. 5. Logging file system maintenance flowchart

4 Conclusion

The logging function designed in this paper can effectively circumvent the black-box characteristics of embedded software, and can record the operation status and debugging information of the software in real time during the operation of the embedded software, which can effectively provide analysis basis for the testers during software testing or problem recurrence, and improve the reliability and usability of the software. The protective design of various boundaries and mechanisms of the logging function also ensures that the function can be called for a long time during the operation of the embedded software without affecting the normal operation of the system, and the function remains transparent to the user level and simple to use for the user.

References

1. Xu Jingfeng. Implementation of real-time event logging in embedded systems [J]. Computer Knowledge and Technology, 2014(24).

2. Liang P, Zhang XL, Chen H, Gu JG. Fault recovery strategy based on real-time logs [J]. Journal of Wuhan University (Science Edition), 2014(02).
3. YANG Yuanyuan,WANG Xiaohua,FENG Sitong,WU Jian. Autonomous logging design for operating system-less electronics[J]. Information Technology and Informatization,2021(05):106-108.
4. Liu Hongwei. Design and implementation of enterprise work log recording software [J]. Heilongjiang Electric Power, 2022,44 (06): 561-564. DOI: https://doi.org/10.13625/j.cnki.hljep.2022.06.018
5. Feng Jie. On the necessity of supervision log record format from the perspective of archiving [J]. Communication and Information Technology, 2017 (05): 54-56
6. Chang Xiaogang, Li Qiang, Wu Ling. Research on PKI-based authentication fraud detection and log recording system [J]. Information Communication, 2016 (08): 103-105
7. Cha Daiyu. Research and Implementation of Embedded Real-Time Memory Database [D]. University of Electronic Science and Technology, 2009.
8. Sheng Yexing. Research on Embedded Database Transaction Processing and Log Recovery Technology [D]. Hefei University of Technology, 2007.
9. Sun, Li-Bin, Zhao, Ming-Ming. Design and implementation of Linux-based embedded logging system[J]. Electronic Science and Technology,2017,04(03):97-99.DOI:https://doi.org/10.16453/j.issn.2095-8595.2017.03.024.
10. Chang S. G., Li Q., Wu L.. Research on PKI-based authentication spoof detection and logging system[J]. Information Communication,2016(08):103-105.
11. Zhang T,Zhang Q. Program design of logging module of TDCS-y dispatching command system[J]. Railroad Computer Application,2015,24(05):47–48+52.