Implementation of Data Handling Feature for Runtime Process Crash

Prajwal G G[1], Shreyas B H[2], Ramya S[3]

[1]Prajwal G G, Dept. of Electronics and Communication Engineering, R V College of Engineering, Bengaluru 560059
[2]Shreyas B H, Dept. of Computer Science Engineering, R V College of Engineering, Bengaluru 560059
[3]Ramya S, Assistant Professor, Dept. of Electronics and Communication Engineering, R V College of Engineering, Bengaluru 560059

---***---

Abstract - *The work carried out in this project consists of multiple processes which runs in parallel. The two processes are user interface process and object management process. These two processes are initialised by the initialise process. User interface process is implemented using glade interface design tool which is based on gtk. The object management process is used to create the objects and signal handler is written for handling SIGTERM signal.*

The results obtained shows that, a total of 10000 objects of class port is created and these objects are written to a text file named sigtermfile.txt once the object management process is terminated using SIGTERM signal from the terminal.

Key Words: **Process termination, Signal Handling, SIGTERM signal, SIGSEGV signal, Glade.**

1. INTRODUCTION

Computer programming has become inevitable part of communication technology. While developing any application software care is taken to avoid the software bugs and logical errors. In spite of testing and improving the coding, bugs in the codes cannot be eliminated by hundred percent. Therefore, process crash may occur due to logical errors or entering invalid data by the user during the run time of the application software.

1.1 Methodology

The initialize process will initialize the user interface process and object management process. The initialize process will monitor the other two processes. The user interface process and object management process are the child processes of initialize process.

The user interface process contains the authentication part, port status change option and service creation option. The service can be created only after the login procedure is completed. The user interface process sends acknowledgement for the initialize process for every ten seconds. This timing of every ten seconds will be generated using SIGALRM signal. This signal is used for generating software interrupt for every ten seconds and the interrupt handler function will send the acknowledgement in the user interface process and SIGALRM interrupt handler will check whether acknowledgement received or not for every ten second in the initialize process.

User interface process and Initialize process will use sockets for sending and receiving acknowledgement. The object management process will create the objects of the port class. The object management process will create 2500 objects of each port type objects and a total of 10000 objects. The object management process will write the objects created during the runtime of the object management process to a text file named sigtermfile.txt. The total number of objects created will be displayed in the terminal during the execution of the object management process. This will help in verifying whether all the objects which are created during the runtime of the object management process are written into text file or not.

1.2 Literature Review

In [1], authors developed a tool called DRACULA. DRACULA detects data races occurred dynamically in signal handlers. Bugs which are harder to detect and reproduce are called race conditions. Race conditions occur due to signals. Signals are software interrupts generated by the Unix systems. The main focus is on data races which caused by signals. Preliminary evaluation techniques and experience using DRACULA are also mentioned.

In [2], a model of crash recovery applied in main memory database has been introduced. This model makes use of the practical features of the main memory database. The primary step in the crash recovery is identifying the first step of the crash such as transaction failure in database. Crash in transaction implies that transaction did not reach its endpoint. Hence, inconsistency is observed in the shadow objects of the database. By executing Rollback operation, the previous consistent state can be recovered.

In [3], authors developed a tool to protect processes from kernel crash. Typical methods used for recovering processes after kernel crash are checking points and recovery box technique. When the main kernel crashed due to a fatal error, the processor will send a non-maskable interrupt to all other processors. Xen Virtual Machine can get the interrupt signal. From this point, XenPR will handle the memory protection using small kernel and jump to the initialization. Then, small core will have the control of operating system.

In [4], Authors have proposed and developed a system which makes use of multiple checkpoints and when the system is about to crash it notices the user and accepts legitimate inputs. User is able to choose between multiple checkpoints. There are different methods to handle software failure or system crash such as restarting the entire systemor restarting the part of the system, but, if the software failure is occurring due to invalid user input or faulty logic of the codes then resting method is not useful.

In [5], System crashes and process crashes are discussed in the context of loris storage stack. Storage stack is the component responsible for the storage of the user data in any computer system. The first step in the process crash is recovering the metadata from the storage stack.

In [6], Authors developed a software tool, which is named as anticrasher. The tool is capable of detecting and notifying the user about the crash. Authors used Flashback technique which allows user to get into previous point of execution. Once the user is notified by the tool about the crash the user will save the unsaved data avoid the accidental data loss due to crash.

In [7], The proposed work first generalizes the fair synchronization problem with respect to concurrent objects. Read or write operations are considered in asynchronous systems. In asynchronous system any number of processes may crash. Then, authors introduced a new failure detector. Authors used it to solve the fair synchronization problem when processes may crash.

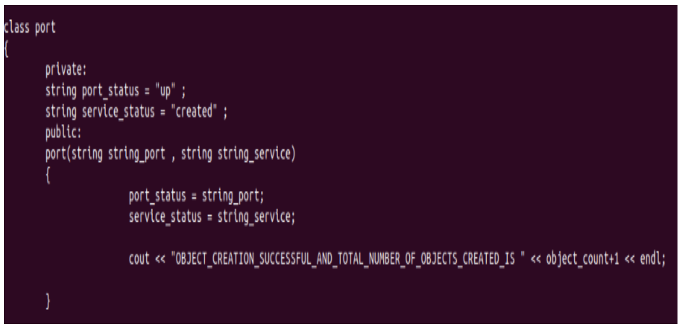
In [8], Authors prioritized crashes based on whether they are security error crashes or safety error crashes. Security error crashes introduce vulnerability to codes and easily exploitable. Authors developed a technique to identify security error crashes using machine learning based on core dump files and last branch record data.

2. Implementation

Object management process consists of class definition of port and this class contains two parameters as its data members. Therefore, the total number of types in which port can be created with the parameters is four. The object management process creates 2500 objects of each type. Therefore, a total of 10000 objects will be created. The constructor of the class definition contains the code which is used for storing the memory address of the created object in an array.

The object management process is written using object-oriented programming style. The object management process contains class definition and also member function which can be called from the main function. Therefore, this function is defined under the public section in the class definition of the port.

The object management process will also contain signal handler function for handling SIGTERM signal. After receiving the SIGTERM signal for the object management process, the object management process will execute the signal handler function. This signal handler function will open a text _le named sigtermfile.txt and writes all the object data which are created during the runtime of the object management process.

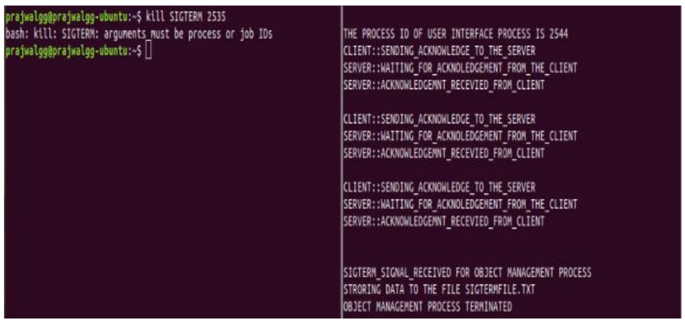


```
class port
{
        private:
        string port_status = "up" ;
        string service_status = "created" ;
        public:
        port(string string_port , string string_service)
        {
                        port_status = string_port;
                        service_status = string_service;

                        cout << "OBJECT_CREATION_SUCCESSFUL_AND_TOTAL_NUMBER_OF_OBJECTS_CREATED_IS " << object_count+1 << endl;

        }
```

Fig -1: Constructor of the class port

Figure 1 shows the constructor of the class port. The figure also shows that there are two data members. The data members are port status and service status. The port status can have two values as up or down. The service status can have two values as created or not created.

The constructor of the class is executed whenever the object of the port class is created. The constructor also displays the total number of objects created till that time in the terminal. This count is used for verifying whether all the objects are written to file or not.



```
prajwalgg@prajwalgg-ubuntu:~$ kill SIGTERM 2535
bash: kill: SIGTERM: arguments must be process or job IDs
prajwalgg@prajwalgg-ubuntu:~$
```

```
THE PROCESS ID OF USER INTERFACE PROCESS IS 2544
CLIENT::SENDING_ACKNOWLEDGE_TO_THE_SERVER
SERVER::WAITING_FOR_ACKNOLEDGEMENT_FROM_THE_CLIENT
SERVER::ACKNOWLEDGEMNT_RECEVIED_FROM_CLIENT

CLIENT::SENDING_ACKNOWLEDGE_TO_THE_SERVER
SERVER::WAITING_FOR_ACKNOLEDGEMENT_FROM_THE_CLIENT
SERVER::ACKNOWLEDGEMNT_RECEVIED_FROM_CLIENT

CLIENT::SENDING_ACKNOWLEDGE_TO_THE_SERVER
SERVER::WAITING_FOR_ACKNOLEDGEMENT_FROM_THE_CLIENT
SERVER::ACKNOWLEDGEMNT_RECEVIED_FROM_CLIENT


SIGTERM_SIGNAL_RECEIVED FOR OBJECT MANAGEMENT PROCESS
STRORING DATA TO THE FILE SIGTERMFILE.TXT
OBJECT MANAGEMENT PROCESS TERMINATED
```

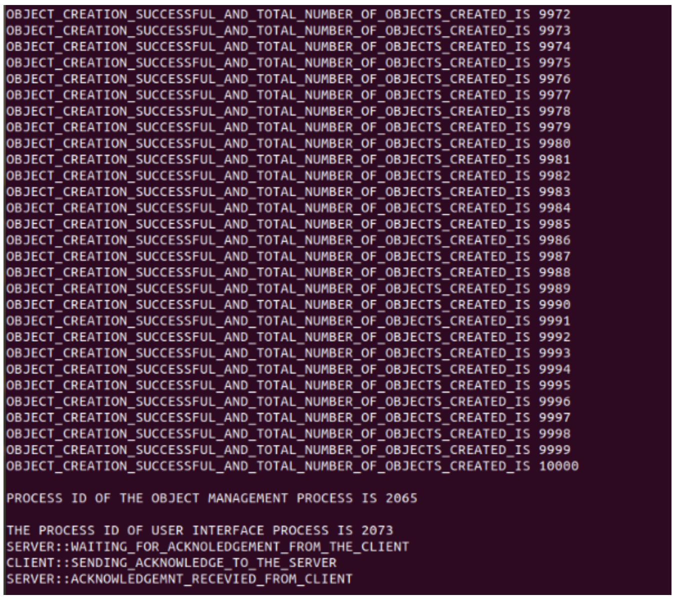
Fig -2: Sending SIGTERM signal for Object Management Process

Figure 2 shows the procedure for sending SIGTERM signal for object management process. This figure consists of two terminals. The on the right consists of running processes and terminal on the left is used to send SIGTERM signal for the object management process with the kill command.

3. Results

The object management process runs in parallel with the user interface process. This process will not interrupt the smooth running of the user interface process. The object management process will create the objects of the class port. The created ports are of four types. The parameters of the created objects differ with respect to port status and service creation status.

Figure 2 shows the output of the object management process. The object management process will create four types of objects. Each object type is created for 2500 times. Hence, a total of 10000 objects are created.

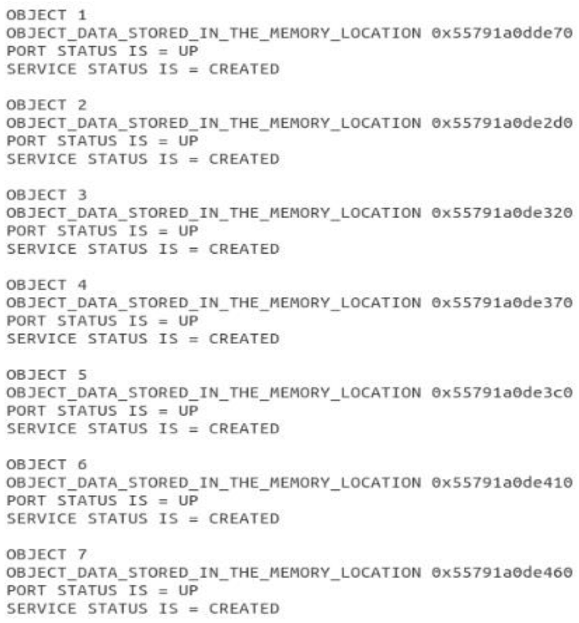


```
OBJECT_CREATION_SUCCESSFUL_AND_TOTAL_NUMBER_OF_OBJECTS_CREATED_IS 9972
OBJECT_CREATION_SUCCESSFUL_AND_TOTAL_NUMBER_OF_OBJECTS_CREATED_IS 9973
OBJECT_CREATION_SUCCESSFUL_AND_TOTAL_NUMBER_OF_OBJECTS_CREATED_IS 9974
OBJECT_CREATION_SUCCESSFUL_AND_TOTAL_NUMBER_OF_OBJECTS_CREATED_IS 9975
OBJECT_CREATION_SUCCESSFUL_AND_TOTAL_NUMBER_OF_OBJECTS_CREATED_IS 9976
OBJECT_CREATION_SUCCESSFUL_AND_TOTAL_NUMBER_OF_OBJECTS_CREATED_IS 9977
OBJECT_CREATION_SUCCESSFUL_AND_TOTAL_NUMBER_OF_OBJECTS_CREATED_IS 9978
OBJECT_CREATION_SUCCESSFUL_AND_TOTAL_NUMBER_OF_OBJECTS_CREATED_IS 9979
OBJECT_CREATION_SUCCESSFUL_AND_TOTAL_NUMBER_OF_OBJECTS_CREATED_IS 9980
OBJECT_CREATION_SUCCESSFUL_AND_TOTAL_NUMBER_OF_OBJECTS_CREATED_IS 9981
OBJECT_CREATION_SUCCESSFUL_AND_TOTAL_NUMBER_OF_OBJECTS_CREATED_IS 9982
OBJECT_CREATION_SUCCESSFUL_AND_TOTAL_NUMBER_OF_OBJECTS_CREATED_IS 9983
OBJECT_CREATION_SUCCESSFUL_AND_TOTAL_NUMBER_OF_OBJECTS_CREATED_IS 9984
OBJECT_CREATION_SUCCESSFUL_AND_TOTAL_NUMBER_OF_OBJECTS_CREATED_IS 9985
OBJECT_CREATION_SUCCESSFUL_AND_TOTAL_NUMBER_OF_OBJECTS_CREATED_IS 9986
OBJECT_CREATION_SUCCESSFUL_AND_TOTAL_NUMBER_OF_OBJECTS_CREATED_IS 9987
OBJECT_CREATION_SUCCESSFUL_AND_TOTAL_NUMBER_OF_OBJECTS_CREATED_IS 9988
OBJECT_CREATION_SUCCESSFUL_AND_TOTAL_NUMBER_OF_OBJECTS_CREATED_IS 9989
OBJECT_CREATION_SUCCESSFUL_AND_TOTAL_NUMBER_OF_OBJECTS_CREATED_IS 9990
OBJECT_CREATION_SUCCESSFUL_AND_TOTAL_NUMBER_OF_OBJECTS_CREATED_IS 9991
OBJECT_CREATION_SUCCESSFUL_AND_TOTAL_NUMBER_OF_OBJECTS_CREATED_IS 9992
OBJECT_CREATION_SUCCESSFUL_AND_TOTAL_NUMBER_OF_OBJECTS_CREATED_IS 9993
OBJECT_CREATION_SUCCESSFUL_AND_TOTAL_NUMBER_OF_OBJECTS_CREATED_IS 9994
OBJECT_CREATION_SUCCESSFUL_AND_TOTAL_NUMBER_OF_OBJECTS_CREATED_IS 9995
OBJECT_CREATION_SUCCESSFUL_AND_TOTAL_NUMBER_OF_OBJECTS_CREATED_IS 9996
OBJECT_CREATION_SUCCESSFUL_AND_TOTAL_NUMBER_OF_OBJECTS_CREATED_IS 9997
OBJECT_CREATION_SUCCESSFUL_AND_TOTAL_NUMBER_OF_OBJECTS_CREATED_IS 9998
OBJECT_CREATION_SUCCESSFUL_AND_TOTAL_NUMBER_OF_OBJECTS_CREATED_IS 9999
OBJECT_CREATION_SUCCESSFUL_AND_TOTAL_NUMBER_OF_OBJECTS_CREATED_IS 10000

PROCESS ID OF THE OBJECT MANAGEMENT PROCESS IS 2065

THE PROCESS ID OF USER INTERFACE PROCESS IS 2073
SERVER::WAITING_FOR_ACKNOLEDGEMENT_FROM_THE_CLIENT
CLIENT::SENDING_ACKNOWLEDGE_TO_THE_SERVER
SERVER::ACKNOWLEDGEMNT_RECEVIED_FROM_CLIENT
```

Fig -3: Creation of objects by Object Management Process

Once the SIGTERM signal is received for the object management process, the process will store the data values of all the 10000 objects which are created during runtime of the object management process. These object values will be stored in the text file named sigtermfile.txt.

The object management process creates 2500 objects of each type and a total of 10000 objects of class port are created during the runtime of the object management process. Therefore, there should be 10000 objects data must be available in the generated text file after receiving the SIGTERM signal for the object management process.

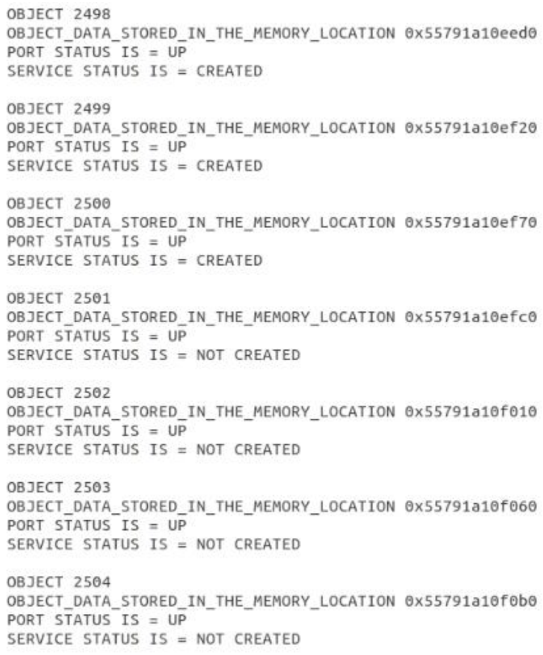


```
OBJECT 1
OBJECT_DATA_STORED_IN_THE_MEMORY_LOCATION 0x55791a0dde70
PORT STATUS IS = UP
SERVICE STATUS IS = CREATED

OBJECT 2
OBJECT_DATA_STORED_IN_THE_MEMORY_LOCATION 0x55791a0de2d0
PORT STATUS IS = UP
SERVICE STATUS IS = CREATED

OBJECT 3
OBJECT_DATA_STORED_IN_THE_MEMORY_LOCATION 0x55791a0de320
PORT STATUS IS = UP
SERVICE STATUS IS = CREATED

OBJECT 4
OBJECT_DATA_STORED_IN_THE_MEMORY_LOCATION 0x55791a0de370
PORT STATUS IS = UP
SERVICE STATUS IS = CREATED

OBJECT 5
OBJECT_DATA_STORED_IN_THE_MEMORY_LOCATION 0x55791a0de3c0
PORT STATUS IS = UP
SERVICE STATUS IS = CREATED

OBJECT 6
OBJECT_DATA_STORED_IN_THE_MEMORY_LOCATION 0x55791a0de410
PORT STATUS IS = UP
SERVICE STATUS IS = CREATED

OBJECT 7
OBJECT_DATA_STORED_IN_THE_MEMORY_LOCATION 0x55791a0de460
PORT STATUS IS = UP
SERVICE STATUS IS = CREATED
```

Fig -4: Text file generated after receiving SIGTERM

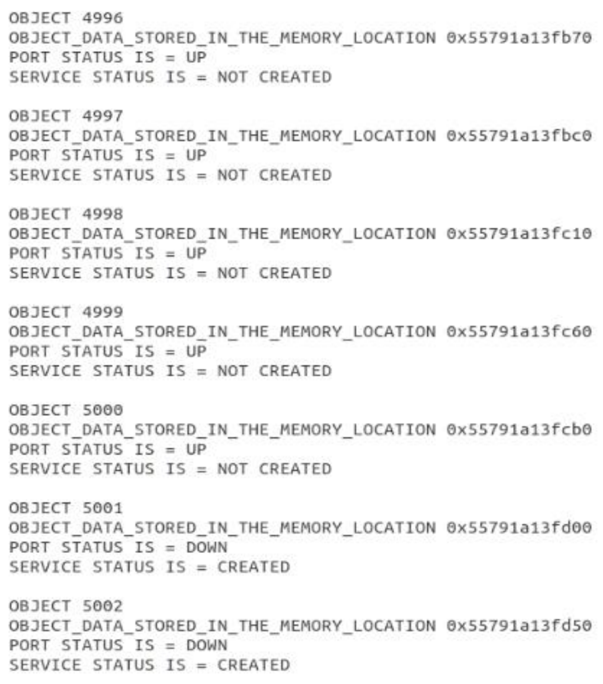
The data in the file shows that for the first 2500 objects the port status is up and service status is created which is same as the parameters with which the objects are created in the object management process.

The data in the file shows that the for the objects from 2500 to 5000 the parameters are up for port status and not created for service status. The same parameters are used to create the objects in the object management process for the objects from 2500 to 5000.



```
OBJECT 2498
OBJECT_DATA_STORED_IN_THE_MEMORY_LOCATION 0x55791a10eed0
PORT STATUS IS = UP
SERVICE STATUS IS = CREATED

OBJECT 2499
OBJECT_DATA_STORED_IN_THE_MEMORY_LOCATION 0x55791a10ef20
PORT STATUS IS = UP
SERVICE STATUS IS = CREATED

OBJECT 2500
OBJECT_DATA_STORED_IN_THE_MEMORY_LOCATION 0x55791a10ef70
PORT STATUS IS = UP
SERVICE STATUS IS = CREATED

OBJECT 2501
OBJECT_DATA_STORED_IN_THE_MEMORY_LOCATION 0x55791a10efc0
PORT STATUS IS = UP
SERVICE STATUS IS = NOT CREATED

OBJECT 2502
OBJECT_DATA_STORED_IN_THE_MEMORY_LOCATION 0x55791a10f010
PORT STATUS IS = UP
SERVICE STATUS IS = NOT CREATED

OBJECT 2503
OBJECT_DATA_STORED_IN_THE_MEMORY_LOCATION 0x55791a10f060
PORT STATUS IS = UP
SERVICE STATUS IS = NOT CREATED

OBJECT 2504
OBJECT_DATA_STORED_IN_THE_MEMORY_LOCATION 0x55791a10f0b0
PORT STATUS IS = UP
SERVICE STATUS IS = NOT CREATED
```

Fig -5: Data stored in the text file

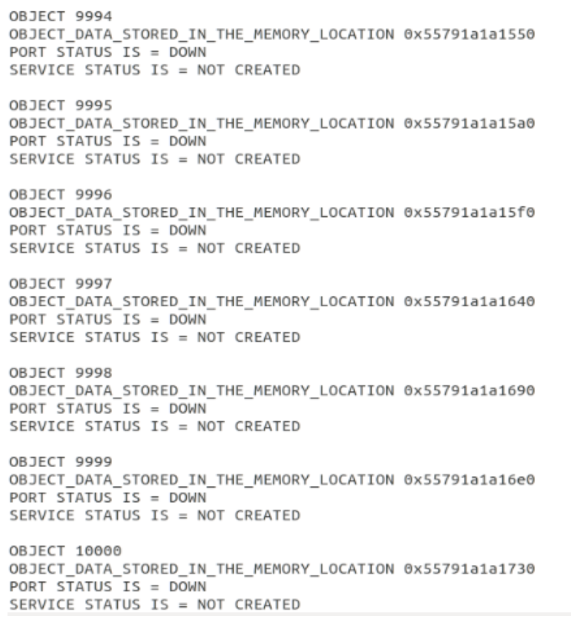


```
OBJECT 4996
OBJECT_DATA_STORED_IN_THE_MEMORY_LOCATION 0x55791a13fb70
PORT STATUS IS = UP
SERVICE STATUS IS = NOT CREATED

OBJECT 4997
OBJECT_DATA_STORED_IN_THE_MEMORY_LOCATION 0x55791a13fbc0
PORT STATUS IS = UP
SERVICE STATUS IS = NOT CREATED

OBJECT 4998
OBJECT_DATA_STORED_IN_THE_MEMORY_LOCATION 0x55791a13fc10
PORT STATUS IS = UP
SERVICE STATUS IS = NOT CREATED

OBJECT 4999
OBJECT_DATA_STORED_IN_THE_MEMORY_LOCATION 0x55791a13fc60
PORT STATUS IS = UP
SERVICE STATUS IS = NOT CREATED

OBJECT 5000
OBJECT_DATA_STORED_IN_THE_MEMORY_LOCATION 0x55791a13fcb0
PORT STATUS IS = UP
SERVICE STATUS IS = NOT CREATED

OBJECT 5001
OBJECT_DATA_STORED_IN_THE_MEMORY_LOCATION 0x55791a13fd00
PORT STATUS IS = DOWN
SERVICE STATUS IS = CREATED

OBJECT 5002
OBJECT_DATA_STORED_IN_THE_MEMORY_LOCATION 0x55791a13fd50
PORT STATUS IS = DOWN
SERVICE STATUS IS = CREATED
```

Fig -6: Data stored in the text file

The data in the file shows that for the objects from 2500 to 5000 the port status is up and service status is not created which is same as the parameters with which the objects are created in the object management process.

The data in the file shows that the for the objects from 5000 to 7500 the parameters are up for port status and not created for service status. The same parameters are used to create the objects in the object management process for the objects from 5000 to 7500.



```
OBJECT 9994
OBJECT_DATA_STORED_IN_THE_MEMORY_LOCATION 0x55791a1a1550
PORT STATUS IS = DOWN
SERVICE STATUS IS = NOT CREATED

OBJECT 9995
OBJECT_DATA_STORED_IN_THE_MEMORY_LOCATION 0x55791a1a15a0
PORT STATUS IS = DOWN
SERVICE STATUS IS = NOT CREATED

OBJECT 9996
OBJECT_DATA_STORED_IN_THE_MEMORY_LOCATION 0x55791a1a15f0
PORT STATUS IS = DOWN
SERVICE STATUS IS = NOT CREATED

OBJECT 9997
OBJECT_DATA_STORED_IN_THE_MEMORY_LOCATION 0x55791a1a1640
PORT STATUS IS = DOWN
SERVICE STATUS IS = NOT CREATED

OBJECT 9998
OBJECT_DATA_STORED_IN_THE_MEMORY_LOCATION 0x55791a1a1690
PORT STATUS IS = DOWN
SERVICE STATUS IS = NOT CREATED

OBJECT 9999
OBJECT_DATA_STORED_IN_THE_MEMORY_LOCATION 0x55791a1a16e0
PORT STATUS IS = DOWN
SERVICE STATUS IS = NOT CREATED

OBJECT 10000
OBJECT_DATA_STORED_IN_THE_MEMORY_LOCATION 0x55791a1a1730
PORT STATUS IS = DOWN
SERVICE STATUS IS = NOT CREATED
```

Fig -7: Total number of objects stored in the text file

4. CONCLUSION

The objects management process has created 10000 objects of class port and these object data values are written into a text file named sigtermfile.txt after receiving SIGTERM signal. In future, the work can be further extended with the more number processes and collecting data of the processes before termination of the process. This technique can be used for collecting data for debug purpose in case of process termination by signal.

REFERENCES

[1] Takamitsu Tahara, Katsuhiko Gondow, Seiya Ohsuga "DRACULA: Detector of Data Races in Signals Handlers" 15th Asia-Pacific Software Engineering Conference, 2008

[2] Tang Yanjun, Luo Wen-hua "A Model of Crash Recovery in Main Memory Database" International Conference On Computer Design And Applications, 2010.

[3] Manbiao Wang, Hao Chen "The Design and Implementation of Process Recovery Mechanism Based on Xen" International Conference on Business Computing and Global Informatization, 2011.

[4] Tsozen Yeh, Weian Cheng "Improving Fault Tolerance through Crash Recovery" International Symposium on Biometrics and Security Technologies, 2012.

[5] David C. van Moolenbroek, Raja Appuswamy, Andrew S. Tanenbaum "Integrated System and Process Crash Recovery in the Loris Storage Stack" IEEE Seventh International Conference on Networking, Architecture, and Storage, 2012.

[6] Anil Kumar Karna, Yuting Chen "Anticrasher: Predicting and Preventing Impending Crashes on Runtime at User End" International Conference on Advances in Computing, Communications and Informatics, 2013.

[7] Carole Delporte-Gallet, Hugues Fauconnier, Michel Raynal "Fair Synchronization in the Presence of Process Crashes and its Weakest Failure Detector" IEEE 33rd International Symposium on Reliable Distributed Systems, 2014.

[8] Shubham Tripathi, Gustavo Grieco, Sanjay Rawat "Exniffer: Learning to Prioritize Crashes by Assessing the Exploitability from Memory Dump" 24th Asia-Pacific Software Engineering Conference, 2017.