DEBUGGING OF PARALLEL PROGRAMS

Marek VYSOKÝ
Department of Computer and Informatics, Faculty of Electrical Engineering and Informatics
Technical University of Košice, Letná 9,042 00 Košice, Slovak Republic, E-mail: vyshko@centrum.cz

SUMMARY

How to debugging programs running in concurrency environment using log files. Types of record in log file. Generating statistic report of using data source, detecting of logical mistakes, detecting of very used data source, detecting errors of recourses allocation by using curve of failed access in critical section. Detecting possible dead lock state from log files converting log files to database and using SQL query.

Keywords: *debugging, parallel, program, log files, resource allocation*

1. INTRODUCTION

We have more different models for design of software products. Phases of design are different in each model, but all models contain phases:

- Analysis
- Implementation
- Debugging
- Testing
- Introducing system
- Maintenance

We need watch correct functionality of designed software product already in phase implementation. In simple one thread systems is possibility connection to instruction and debugging program instruction by instruction, using good or worst debugger programs.

Parallel systems running in concurrency environment have many threads and there are problems connecting thread's instructions. Problems are following:

- Debugging of instruction had overhead. Start and stop of instruction take time and during this time could make changes by other thread.
- It is impossible manage debugging of several threads and result can be out of focus.

[1]

In this case we have to choose other way, wherein overheads of debugging are minimal and result is not out of focus.

Adequate way debugging of parallel programs is creating log files. We can describe this as inserting information about rise event to log file.

To achieve required result, we have to reckon with log files already in first phases of design. Investment to development of logging facilities is very important, because quality design logging facilities provides simply reusability and retrieving of investment.

2. LOG RECORD

Type of log record is depend on type of debugging data

Minimal record should be involved:
[**date and time**] [**source**] [**type of event**] [**event**] [**additional information**]

Example:
[24.10.200314:30:001][Proces1][information] [open file][c:\abcd.txt]
[24.10.2003 14:30:001][Proces2][error][file not found][c:\abcd.txt]

Types of events are possible divide to several categories
In practice are often using following categories:

- Information – event, which had been occurred in system, but not had affect on continuity of system and is used to monitoring of event.

[24.10.2003 14:30:001][Proces1][information][open file][c:\abcd.txt]

- Warning – event, which had been occurred and had affect on continuity of system but not evoked failure of executed operation.

[25.10.2003 17:10:135][s1][warning][count of login achieved maximum]

- Error – event, which had been occurred and had affect on continuity of system and evoked failure of executed operation

[26.10.2003 10:16:555][a6][error][object has null value]

- Critical error – event, which had been occurred and had affect on continuity of system and evoked failure of executed operation and program

[26.10.2003 10:16:555][z3][error][memory is not available]

In object oriented design of log facilities we need design simple configurable and reusable facilities. We have many reasons to debugging of parallel processes.

3. DETECTING OF LOGICAL MISTAKE

To detecting logical mistake we are need implementing methods of conditional writing to log file. It's mean that apart from standard input parameters to log file record, it is extending by next parameter and that is condition of writing. There are useful implementing methods of positive writing, where record is inserting in log file if condition is true and negative writing where record is inserting to log file if condition is false. Combination of conditional writing and standard writing is possible detecting of logical mistakes and exceptions. We able to watching variable by using of additional information, but in this case are disadvantaging that log file can growing up. To eliminate many not important record we can configured log facilities by config file, which setting up level and range of log records. In practice are using xml document, which application reads on start and log facilities decide by setting in xml, which record will be writing.

4. DETECTING OF CONCURRENCY FOR SHARED SOURCES

On parallel processes rise problems of concurrency for shared sources (files, databases, hardware...).

Critical section – rises in concurrency for one shared source. Critical in this case are:

- Process overload source and do not able access to source for other processes.
- Processes are reading/writing data by themselves without commit (dirty read/write)

```
01  read ZA
02  ZA=ZA+1
03  write ZA
```

This small program, which counts of access to source ZA has been executed by two parallel threads can evocate dirty read and write in case, that:
ZA has in source value 0
Process 1 and 2 executed program in order:

```
P1 execute          P2 execute
01  ZA=0

02  ZA=ZA+1
                    01  ZA=0
03  ZA=1
                    02  ZA=ZA+1

                    03  ZA=1
```

After executed programs by processes is certain that counter ZA in source shows 1, but there was two access to source ZA.

Concept of allocated source is very difficult area and using many of methodologies and algorithms for example semaphores, transaction atc.
[3]

In implementation of difficult processing of data is useful recording and detecting success in concurrency of processes to shared source. In object oriented design we are extending methods for record input and output into and from critical section. In this case is important recording success of using source and time of staying in critical section. Record is following:

[Datetime][Process][Section:Source][Input/Output] [(No)Succes]

Creating couple of input and output we are getting time of stay in critical section and getting information about success of source allocation.

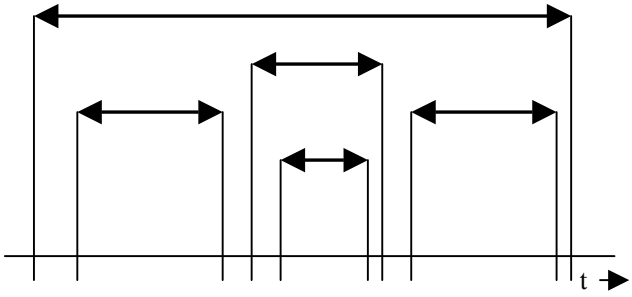

Fig. 1 Input and output from critical section

We have to transform records from log file into table. This table is representation time of abidance in critical section.

```
CREATE TABLE SECTION
(
        ID int,
        DtInput datetime,
        DtOutput datetime
        Process varchar(20),
        Source varchar(20),
        Success varchar(1)
)
```

Then for each record in table SECTION we are generating count of parallel request on source in critical section during abidance of process in critical section.

Query is following:

In first step we load parameters from record for which we are finding count of parallel input.

For all IDs in table Section make

Select @dtInput= DtInput, @dtOutput=DtOutput, @Process =Process,

@Source=Source from SECTION where ID=@ID

Then select counts for this input

```
Select @C=count(*) from SECTION
Where ID!=@ID AND Source=@Source
          AND NOT
          ( (DtInput<= @DtInput
          AND
          DtOutput <= @DtOutput)
          OR
          (DtInput>=@DtInput
           AND
          DtOutput>=@DtOutput)
           )
```

this sequence we are using for each record in table SECTION

Curve of access $Ca(t)$ – are counts of requests on source in time interval

Curve of satisfied access $Csa(t)$ (fig. 2) – are counts of requests, which finish with success of allocating of source. It's created by add condition in query 1
Success='Y'

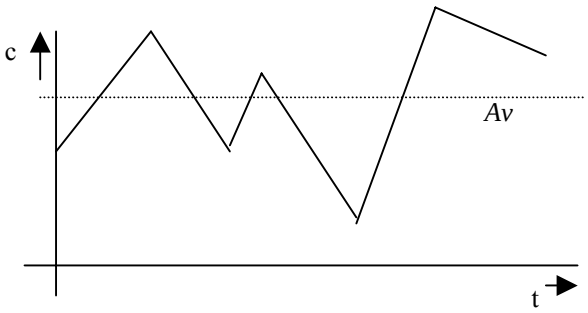

Fig. 2 Curve of satisfied access

Curve of not satisfied access $Cnsa(t)$ – are counts of requests, which finish with failure of allocating of source. It's created by add condition in query 1
Success='N'

$P(t)$ – Count of changes in time t

Av – Average value of request on source in critical section

$$Av = \frac{\sum Ca(t)}{P(t)} \tag{1}$$

Avn – Average value of not satisfied request on source in critical section

$$Avn = \frac{\sum Cnsa(t)}{P(t)} \tag{2}$$

5. DETECTING ERRORS IN RESOURCE ALLOCATION

Problems with allocating source in critical section have different rise.

- Source is not able to assigning data (fig. 3).
  This state is finding if average value of success access are equal to zero. It's mean:

$$\lim_{t->\infty} Ca(t) = \lim_{t->\infty} Cnsa(t)$$
and
$$\lim_{t->\infty} Csa(t) = 0 \tag{3}$$

  and second additional information is that in log file exist errors of failure source handling.

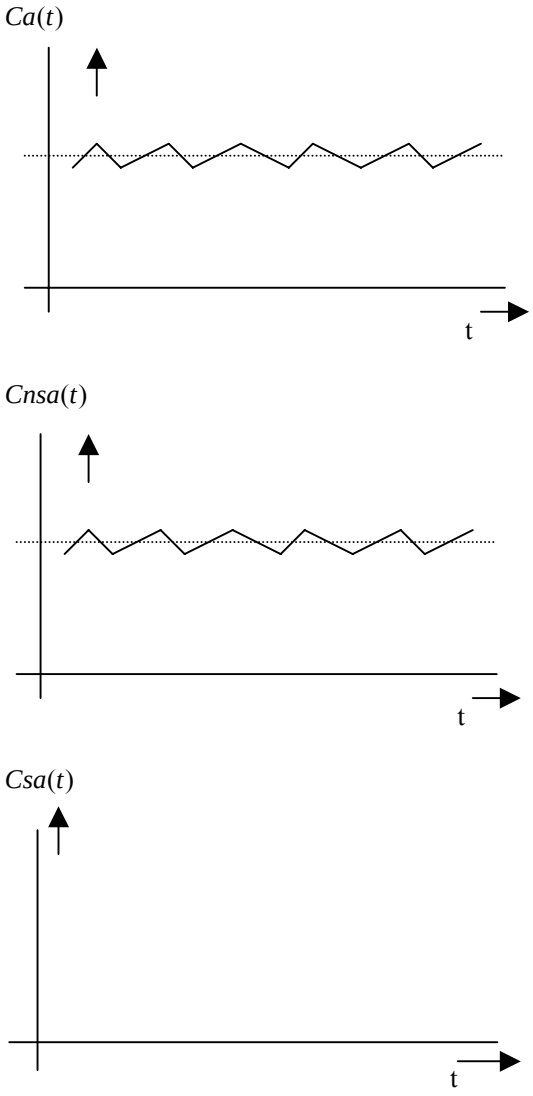

Fig. 3 Example for state source is not able assign data

- Source is overloading by one process (fig. 4), or assigning technique is too slow.
  This state is finding if average value of success access is too low.

$$\lim_{t->\infty} Ca(t) = Ava$$
and
$$\lim_{t->\infty} Cnsa(t) = Avns \quad (4)$$
and
$$\lim_{t->\infty} Csa(t) = Avs$$
and
$$Avs \le Avns \le Ava$$

When $Avs = 1$ then source is blocking by one process.

When Avs is too low or $Avns$ is too high then assigning technique for resource is too slow.

.

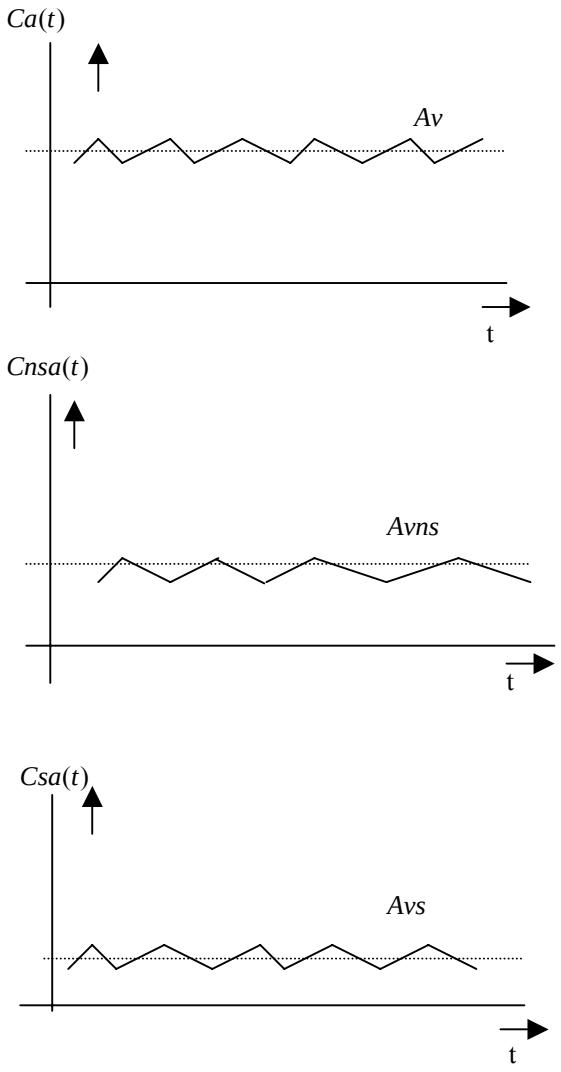

Fig. 4 Example for source is overloading by one process

6. DETECTING DEADLOCK STATE

What is dead lock state in system? It's occurs that one process lock access to one source and then need access to other source data, which is locked by other process and this process is waiting to data from source locked by first process. Both processes are waiting to unlock source and access to data failed. [2]

Example of programs which can produce deadlock:

Process1	Process2
Read Z1	Read Z2
Lock Z1	**Lock Z2**
Read Z2	**Read Z1**
Z1=Z2+Z1	Z2=Z1+Z2
Write Z1	Write Z2
UnLock Z1	Unlock Z2

[5]

We insert following record for request to source data

[Datetime][Process][Source][Type]
[24.10.200314:30:001][process1][source1][requeststart]

and for finishing request to source create record

[Datetime][Process][Source][Type]
[24.10.200314:30:031][process1][source1][requeststop]

we can transform log file to data table by creating table

```
REQUEST
(                                   Sn
        ID int,
        DtInput datetime,
        DtOutput datetime
        Process varchar(20),
        Source varchar(20)
)
```

we are inserting request, it is coupling records from log file where DtInput is when type=requeststart and DtOutput is when type=requeststop for same source and process.

For detecting deadlock state we are need inserting record with following information about lock source.

[Datetime][Process][Source][Lock]
[24.10.2003 14:30:001][process 1][source 1][lock]

and for unlock source insert record :

[Datetime][Process][Source][Unlock]
[24.10.200314:30:121][process 1][source 1][unlock]

we can transform log file to table

```
CREATE TABLE LOCK
(
        ID int,
        DtInput datetime,
        DtOutput datetime
        Process varchar(20),
        Source varchar(20)
)
```

Insert coupling records for lock and unlock source. Detecting of deadlock state is following:
In first step load parameter for all records in table Lock

```
Select @dtInput= DtInput , @dtOutput=DtOutput,
@Process =Process,
      @Source=Source from LOCK  where ID=@ID
```

create temporary table where detect couple of input and output to critical section with query

```
Select Process,Source INTO #tmp_request from
REQUEST
Where Source=@Source
          AND NOT
          ( (DtInput<= @DtInput
          AND DtOutput <= @DtOutput)

          OR

          (DtInput>=@DtInput
           AND DtOutput>=@DtOutput)
                )
```

and lock temporary table

```
Select Process,Source INTO #tmp_lock from
LOCK
Where Source=@Source
          AND
          NOT
          ( (DtInput<= @DtInput
          AND DtOutput <= @DtOutput)
          OR
          (DtInput>=@DtInput
          AND
          DtOutput>=@DtOutput)
            )
```

and create table #tmp_lock_request with query

```
Select l.Process ProcessLock,r.Process
ProcessRequest, Source
INTO #tmp_lock_request
          from #tmp_lock
join #tmp_request r on  l.Source=r.Source
```

and deadlock is detecting by query

```
select * from #tmp_lock_request  t1
   join #tmp_lock_request t2
          on t1.LockProcess=t2.RequestProcess
          AND
          t1.RequestProcess=t2.LockProcess
```

by this technique we are detecting state, where one process lock data for other process and contrariwise. In this case both process finish with not satisfied result.

7. CONCLUSION

During design and during introducing phase is necessarily ensuring consistency and right functionality of software product. When we are making investments to logging facilities it is probably that we are able to take advantage of it in introducing and maintenance software product. We are able to configuring logging facilities for detecting deadlocks and we are able to provide statistical monitoring of using data sources. Next advantage of logging facilities is that when programs rising error or exception we are able to detecting place and reason of error and make correction of problem. Good service is ability of monitoring count of users currently using resource and count of handling users at time. We able to growing up hardware facilities or managing database indexing in this case. Other view on log facilities is security aspect. It is powerful utility to monitoring security incident of users. We are able to monitoring access to each data sources and detect and prevent not authorized access and diversion of data sources.

REFERENCIES

[1] Performance Measurement and Debugging http://www.lindaspaces.com/book/chap4.htm, 2005
[2] Ben Adida, Detecting and Breaking Deadlock http://web.mit.edu/6.033/1997/reports/r03-ben.html. 2005
[3] A Classic Problem - Dining Philosophers www.isi.edu/~faber/cs402/notes/lecture8.pdf 2005
[4] how to detect deadlock? www.soe.ucsc.edu/classes/ cmps111/Fall04/Slides/cmps111-chap3.pdf
[5] Pun H. Shiu, Yudong Yudong Tan,Vincent J. Mooney, A Novel Parallel Deadlock Detection Algorithm and Architecture, Georgia Tech, 2001

BIOGRAPHY

Marek Vysoký was born in 1978. He graduated (Ing.) with distinction at the Faculty of Electrical Engineering and Informatics at Technical University in Košice in 2001. He is external PhD. Student at Department of Computer and Informatics at Technical University in Košice. He deals with security informatics systems.