Real Alternative DBMS ALTIBASE, Since 1999

문제분석을 위한 OS별 유틸리티 사용 가이드

2010. 02

Document Control

Change Record

Date	Author	Change Reference
2010-02-24	lim272	Created

Reviews

Date	Name (Position)

Distribution

Name	Location

목차

개요

DBMS와 연관된 기술 지원을 하게 되면 ALTIBASE가 제공하는 성능뷰만으로는 문제 해결을 위한 정보가 부족한 경우들이 있다. 이때 운영체제에서 제공하는 몇 가지 명령어를 이용하여 필요한 정보를 획득할 수 있는데 본 문서에서 그와 관련된 명령어들을 설명한다.

공통 명령어

모든 운영체제에 실행이 가능한 특별히 제약을 갖지 않는 공통적인 명령어를 설명한다.

netstat

네트웍 설정이나 오류 패킷 여부가 존재하는지 확인한다.

Shell> netstat –in

Name	Mtu	Net/Dest	Address	Ipkts	Ierrs	Opkts	Oerrs	Collis	Queue
lo0	8232	127.0.0.0	127.0.0.1	2314073378	0	2314073378	0	0	0
ce0	1500	192.168.1.0	192.168.1.60	501973072	0	1690153823	0	0	0

패킷상의 송수신간에 문제가 발생할 경우 Ierrs/Oerrs/Collis 등이 수치가 증가된다. 이 경우는 네트웍상에 어떤 문제가 있을 수 있음으로 관리자로 하여금 점검하도록 조치한다.

vmstat

시스템의 메모리, 디스크, swap in/out, CPU의 상태 등을 전반적으로 사용자가 정한 간격으로 조회할 수 있다.

Shell> vmstat 1 5 (1 초단위로 5 번을 화면에 출력)

```
kthr      memory            page            disk          faults      cpu
r b w   swap  free  re  mf pi po fr de sr m1 rm s6 sd   in   sy   cs us sy id
4 0 253 18600904 3703904 471 6104 110 36 39 0 10 25 -0 -0 -9 15845 92177 31495 24 17 60
4 0 2098 11390376 2018056 622 10532 155 15 15 0 0 3 0 0 0 23252 46363 40809 34 24 42
0 0 2098 11390376 2019160 622 11288 16 88 88 0 0 5 0 0 0 22014 67475 39977 33 24 44
0 0 2098 11390376 2019416 667 11337 0 8 8 0 0 2 0 0 0 22958 175682 40640 31 27 42
0 2 2098 11390280 2019840 691 11385 0 88 80 0 0 39 0 0 0 22573 185484 41354 33 28 39
```

확인할 주요 지표는 다음과 같다.

항목	설명
Kthr의 r	CPU를 점유하기 위해 대기하는 쓰레드의 개수이며 값이 크다면 CPU병목이 발생한다고 판단할 수 있다.
Memory의 free	물리적 메모리상의 여유 공간
Page의 fr, sr	Fr, sr의 수치가 증가 한다는 것은 메모리 부족에 의해 메모리를 확보하기 위한 victim 페이지에 대한 scan 및 디스크I/O가 발생한다는 의미이다.
CPU의 모든 항목	각 항목별로 변화 추이를 살펴야 한다.

SUN

prstat

ALTIBASE는 쓰레드 구조로 개발되어 있으며 경우에 따라 어떤 쓰레드가 CPU를 많이 점유하고 사용하는지 확인해야 하는 경우 사용한다.

Shell> **prstat –L –p <process id> <refresh interval>**

Ex) prstat –L –p 22951 1 (22951 프로세스를 1 초 단위로 보겠다는 의미)

PID	USERNAME	SIZE	RSS	STATE	PRI	NICE	TIME	CPU	PROCESS/LWPID
22951	lim272	502M	106M	sleep	59	0	0:22:50	1.7%	altibase/5
22951	lim272	502M	106M	sleep	59	0	0:23:49	0.3%	altibase/4
22951	lim272	502M	106M	sleep	59	0	0:00:10	0.1%	altibase/12
22951	lim272	502M	106M	sleep	59	0	0:00:05	0.1%	altibase/47
22951	lim272	502M	106M	sleep	59	0	0:25:28	0.0%	altibase/6
22951	lim272	502M	106M	sleep	59	0	0:22:19	0.0%	altibase/9
22951	lim272	502M	106M	sleep	59	0	0:23:13	0.0%	altibase/8

항목	설명
CPU	해당 쓰레드가 CPU를 현재 사용하는 CPU 점유율
LWPID	쓰레드의 고유번호

pstack

prstat과 같이 쓰레드의 CPU점유를 확인하면 쓰레드가 현재 어떤 부분을 수행 중인지 확인하는 방법으로 사용한다.

Shell> **pstack –F <process pid> | c++filt**

---------- lwp# 7 / thread# 7 ----------

ffffffff7e1d9ce8 pollsys (ffffffff7b4ffa50, 0, ffffffff7b4ffb10, 0)

ffffffff7e173c44 pselect (0, ffffffff7b4ffa50, ffffffff7e344710, ffffffff7e344710, ffffffff7b4ffb10, 0) + 1f0

ffffffff7e173fe8 select (0, 106eea070, 0, 0, ffffffff7b4ffbd8, fffc00) + a0

00000001006f6df4 IDE_RC cmnDispatcherSelectSOCK(cmnDispatcher*,iduList*,unsigned*,PDL_Time_Value*) (106eea040, ffffffff7b4ffd88, 0, 106cc5818, fffffff, 106eea070) + 50

00000001006f089c IDE_RC

cmiSelectDispatcher(cmnDispatcher*,iduList*,unsigned*,PDL_Time_Value*) (106eea040, ffffffff7b4ffd88, 0, 106cc5818, ffffffffffffffff, 1006f6da4) + 1c

0000000100140790 void mmtServiceThread::findReadyTask() (106cc56d8, 105966f18, 101000, 0, 2710, 0) + 14

000000010013f878 void mmtServiceThread::run() (106cc56d8, 1, ffffffffffffffff, 2ec6bb9, 101329000, 10132b000) + 560

0000000100717034 void*idtBaseThread::staticRunner(void*) (106cc56d8, 100000, 0, 0, 100a120c8, 10013f318) + 14

ffffffff7e1d609c _lwp_start (0, 0, 0, 0, 0, 0)

---------- lwp# 8 / thread# 8 ----------

000000010056581c IDE_RC smlLockMgr::lockTable(int,smlLockItem*,smlLockMode,unsigned long,smlLockMode*,idBool*,smlLockNode**,smlLockSlot**) (11b2176c0, 102e8cd50, 200, 7, 11644ea00, 7)

000000010067ac5c IDE_RC smiTableCursor::open(smiStatement*,const void*,const void*,unsigned long,const smiColumnList*,const smiRange*,const smiRange*,const smiCallBack*,unsigned,smiCursorType,smiCursorProperties*) (11b2176c0, ffffffff7b2fdc70, 11644e9e8, 0, 155cec, 0) + 38c

00000001001e6f2c IDE_RC qmcInsertCursor::openCursor(smiStatement*,unsigned,smiCursorProperties*) (ffffffff7b2fdb18, ffffffff7b2fdc70, 1, ffffffff7b2fdb38, 0, 0) + 64

000000010023e0a8 IDE_RC qmx::executeInsertValues(qcStatement*) (106ea13c8, 106ea14b8, 11b1e81f8, 6200, 0, 10023de38) + 270

0000000100269cec IDE_RC qsxExecutor::execNonSelectDML(qsxExecutorInfo*,qcStatement*,qsProcStmts*) (ffffffff7b2fec78, 106ea13c8, 10469b2c8, 106cb1978, 1000000000, 1) + 3f0

00000001002698a8 IDE_RC qsxExecutor::execInsert(qsxExecutorInfo*,qcStatement*,qsProcStmts*) (ffffffff7b2fec78, 106ea13c8, 10469b2c8, 1, 101135000, ffffffffffffffff) + c

0000000100268344 IDE_RC qsxExecutor::execStmtList(qsxExecutorInfo*,qcStatement*,qsProcStmts*,idBool) (ffffffff7b2fec78, 106ea13c8, 10469b2c8, 100268290, 0, 11b2117e0) + 6c

000000010026b914 IDE_RC qsxExecutor::execFor(qsxExecutorInfo*,qcStatement*,qsProcStmts*) (ffffffff7b2fec78, 106ea13c8, 11b1df798, 1, 106ef0b88, 1) + 58c

0000000100268344 IDE_RC qsxExecutor::execStmtList(qsxExecutorInfo*,qcStatement*,qsProcStmts*,idBool) (ffffffff7b2fec78, 106ea13c8, 11b1df798, 100268290, 0, 8080808080808080) + 6c

0000000100267da8 IDE_RC qsxExecutor::execBlock(qsxExecutorInfo*,qcStatement*,qsProcStmts*) (ffffffff7b2fec78, 106ea13c8, 11b231478, 0, 0, 106ef0b88) + d4

0000000100267054 IDE_RC qsxExecutor::execPlan(qsxExecutorInfo*,qcStatement*,mtcStack*,int,qmcCursor*,qmcdTempTableMgr*) (ffffffff7b2fec78, 106ea13c8, 11b211840, 106ea14b8, ffffffff7b2fede0, 1) + 348

0000000100262ecc IDE_RC qsx::callProcWithStack(qcStatement*,qsProcParseTree*,mtcStack*,int,qmcCursor*,qmcdTempTableMgr*,qcTemplate*) (106ea13c8, 11b231518, 11b1fe458, 3ff, ffffffff7b2fede0, ffffffff7b2fedc8) + 78

000000010026299c IDE_RC qsx::executeProcOrFunc(qcStatement*) (106ea13c8, 0, 28, 101225, 0, 0)

+ 1e0

0000000100169528 IDE_RC qci::execute(qciStatement*,smiStatement*) (106ea13c8, 0, 34247000, 28, 4, 100a12f7c) + 2d8

000000010015bfc0 IDE_RC mmcStatement::executeSP(mmcStatement*,long*) (106ea0db8, ffffffff7b2ffcd0, 0, 0, 34249000, ffffffffffffffff) + c

0000000100158cec IDE_RC mmcStatement::execute(long*) (100a12000, ffffffff7b2ffcd0, 100a12, 106ea0db8, 20, 101325000) + bc

0000000100146f28 IDE_RC doExecute(mmtCmsExecuteContext*) (ffffffff7b2ffcc0, 106cb16c8, 1, ffffffff7b2ffcd0, 106ea13c8, 2) + 40

00000001001479ac IDE_RC mmtServiceThread::executeProtocol(cmiProtocolContext*,cmpProtocol*,void*,void*) (106e28940, 106e28990, 106e28900, 106cc5580, 1a, 7fffffff) + c4

00000001006f1ec0 IDE_RC cmiReadProtocolAndCallback(cmiProtocolContext*,void*,PDL_Time_Value*) (106e28940, 106cc5580, 0, 101205ed8, 0, 0) + 90

000000010013f4ec void mmtServiceThread::run() (106cc5580, 1, 106e28940, 101325000, 2400, 10132b000) + 1d4

0000000100717034 void*idtBaseThread::staticRunner(void*) (106cc5580, 100000, 0, 0, 100a120c8, 10013f318) + 14

ffffffff7e1d609c _lwp_start (0, 0, 0, 0, 0, 0)

c++Filter 라는 명령은 C/C++간에 호출된 함수 명이 제대로 보여 지지 않는 경우를 제거하는 용도이다. 사용하지 않을 경우 함수 명이 보기 힘든 형태로 출력됨으로 가능한 사용하도록 하며 일반적으로 컴파일러가 설치된 경로의 실행 파일이 위치하는 곳에 존재한다. (Ex: /opt/SUNWspro/bin/)

위 결과는 모두 쓰레드 별로 출력되기 때문에 각 쓰레드 별로 구분하고 단락 내에서는 가장 아래부터 위로 확인한다.

위의 예제에서 LWP#8 번은 다음과 같은 순서이다.

항목	설명
Lwp_start staticRunner run	쓰레드의 생성 및 개시
cmiReadProtocolAndCallback	통신상에서 사용자의 질의 요청을 읽었음
mmtServiceThread::executeProtocol doExecute	실행 단계로 진입
mmcStatement::execute(long **mmcStatement::executeSP** qci::execute **qsx::executeProcOrFunc** qsx::callProcWithStack	프로시져 수행 요구임을 확인

qsxExecutor::execBlock qsxExecutor::execStmtList **qsxExecutor::execInsert** **qsxExecutor::execNonSelectDML**	해당 프로시져 내에서 Insert문 수행을 확인
qmx::executeInsertValues	Insert문이 실행
qmcInsertCursor::openCursor smiTableCursor::open	MVCC와 관련된 내부 커서를 열음
smlLockMgr::lockTable	테이블에 Lock을 획득

위의 결과와 같이 prstat/pstack의 정보를 조합하면 CPU를 가장 많이 사용하는 특정 쓰레드가 어떤 일을 수행 중인지 확인이 가능하다. 또한, 많은 SQL문중에 위의 pstack결과를 통해 좀 더 범위를 좁힌 추적을 가능하게 할 수 있다.

pfiles

pfiles는 프로세스가 사용 중인 모든 파일 목록을 보여 준다.

```
Shell> pfiles –F <process id>
22951:    /home2/lim272/work/altibase_home/bin/altibase -p boot from admin
Current rlimit: 65535 file descriptors
0: S_IFREG mode:0644 dev:118,38 ino:3309572 uid:124 gid:1 size:202128
O_WRONLY|O_APPEND|O_CREAT|O_LARGEFILE
/home2/lim272/work/altibase_home/trc/altibase_boot.log

1: S_IFREG mode:0644 dev:118,38 ino:3309573 uid:124 gid:1 size:1413256
O_WRONLY|O_APPEND|O_CREAT|O_LARGEFILE
/home2/lim272/work/altibase_home/trc/altibase_sm.log
```

위의 결과에서 현재 사용 가능한 file descriptor정보와 함께 0 번, 1 번 순으로 해당 프로세스가 접근하여 사용 중인 모든 파일들을 보여 준다.

여러 세션들이 접속된 상태에서 pfiles를 수행하면 데이터파일, 트레이스 로그, 트랜잭션 로그파일 외에도 통신 연결 목록까지 모두 보여 준다.

```
321: S_IFSOCK mode:0666 dev:329,0 ino:63732 uid:0 gid:0 size:0
O_RDWR|O_NONBLOCK
SOCK_STREAM
SO_REUSEADDR,SO_KEEPALIVE,SO_SNDBUF(65536),SO_RCVBUF(32788),IP_NEXTHOP(0.0.128.20)
sockname: AF_INET 127.0.0.1  port: 27584
```

peername: AF_INET 127.0.0.1 port: 42567

시스템 로그

기술 지원 시 간혹 외부에서 원인을 찾아보아야 할 경우 SUN은 /var/adm/messages.* 파일들을 확인하도록 한다. 파일의 확장자는 로그가 기록된 주를 의미하며 오늘을 포함한 주간의 로그는 messages 파일에 기록된다.

Shell> vi /var/adm/messages
Feb 24 18:08:24 v880 Corrupt label; wrong magic number
Feb 24 18:08:24 v880 scsi: [ID 107833 kern.warning] WARNING: /pci@9,700000/fibre-channel@4/fp@0,0/ssd@w210000d023041a42,7 (ssd13):
Feb 24 18:08:24 v880 Corrupt label; wrong magic number
Feb 24 18:08:24 v880 scsi: [ID 107833 kern.warning] WARNING: /pci@9,700000/fibre-channel@4/fp@0,0/ssd@w210000d023041a42,7 (ssd13):
Feb 24 18:08:24 v880 Corrupt label; wrong magic number

시스템 로그는 각 벤더의 전문가가 아닌 이상 명확하게 이해하기 어렵지만 장애 등으로 기술 지원을 할 때 특정 시각에 의미 있는 로그가 있다면 반드시 확인하도록 한다.

AIX

AIX 5.1 이전에서는 특정 명령어들은 지원되지 않을 수 있다.

ps

SUN의 prstat과 동일한 결과를 확인할 수 있다.

```
Shell> ps –mo THREAD –p <process id>
USER       PID      PPID      TID ST   CP  PRI  SC    WCHAN  F  TT BND COMMAND
lim272  757988         1        - A   119   60 35        40001      -   -  lim272/work/alti
-          -          -          1540169 S     0    60   1 f100070f10017840 8c10400      -   - -
-          -          -          1708137 S     0    60   1        -    410410      -   - -
-          -          -          1749229 S     0    60   1        -    410410      -   - -
-          -          -          1867981 S    39    79   1        -    418410      -   - -
```

위 결과에서는 쓰레드가 사용하는 점유율은 CP 칼럼으로 확인하면 된다.

procstack

SUN의 pstack과 동일한 결과를 확인할 수 있다.

```
Shell> procstack <process id>
---------- tid# 6901809 (pthread ID:       258) ----------
0x0900000000062a14    write(??, ??, ??) + 0x1c8
0x00000001000b9a60
cmnSockSend(cmbBlock*,cmnLinkPeer*,int,PDL_Time_Value*,idvStatIndex)() + 0x308
0x00000001000b8d38    cmnLinkPeerSendTCP(cmnLinkPeer*,cmbBlock*)() + 0x30
0x000000010007f28c    cmiWriteBlock(cmiProtocolContext*,idBool)() + 0x24c
0x000000010007ccc0    cmiFlushProtocol(cmiProtocolContext*,idBool)() + 0xa8
0x00000001000cd838    mmtServiceThread::executeTask()() + 0xc1c
0x00000001000cbb80    mmtServiceThread::multiplexingAsShared()() + 0x84
0x00000001000cc594    mmtServiceThread::run()() + 0x4c4
0x0000000100077bd4    idtBaseThread::staticRunner(void*)() + 0x28
0x09000000004a44f4    _pthread_body(??) + 0xdc
```

Pstack의 결과를 해석하는 방법과 동일하게 쓰레드 별로 tid#을 기준으로 단락 단위로 분리하여 아래에서 위로 해석해 가도록 한다. 위 결과에서는 어떤 질의가 수행된 이후 결과에 대한 통신 쓰레드의 송신 부분이 기록되어 있음을 확인할 수 있다.

procfiles

SUN의 pfiles와 동일한 결과를 보여 준다

Shell> pfiles –n <process id>

757988 : /home2/lim272/work/altibase_home/bin/altibase -p boot from admin

Current rlimit: 100 file descriptors

0: S_IFREG mode:0200 dev:53,1 ino:2731329 uid:222 gid:1 rdev:0,0 O_WRONLY | O_APPEND
size:451248 name:/home2/lim272/work/altibase_home/trc/altibase_boot.log

1: S_IFREG mode:0222 dev:53,1 ino:2731337 uid:222 gid:1 rdev:0,0 O_WRONLY | O_APPEND
size:3014040 name:/home2/lim272/work/altibase_home/trc/altibase_sm.log

-n옵션을 써야 사용하는 파일명까지 확인할 수 있다.

errpt

운영 장비에 오류가 발생했는지 시스템 로그를 확인하기 위해 사용한다. 디스크 장치의 오류나 네트웍 장치의 오류 혹은 프로세스의 비정상적인 종료 등에 대한 로그를 확인할 수 있기 때문에 기술 지원 시 간혹 외부에서 원인을 찾아보아야 할 경우 반드시 시스템 로그를 확인하도록 해야 한다.

Shell> errpt –a | more

LABEL: CORE_DUMP

IDENTIFIER: C69F5C9B

Date/Time: Thu Feb 25 03:59:12 KORST 2010

Sequence Number: 23893

Machine Id: 00C76BFD4C00

Node Id: aix53-p5

Class: S

Type: PERM

Resource Name: SYSPROC

Description

SOFTWARE PROGRAM ABNORMALLY TERMINATED

Probable Causes

SOFTWARE PROGRAM

HP-UX

HP CPU의 종류에 따라 PA-RISK/ITANIUM으로 분리되는데 PA-RISK장비에서 일부 명령어는 지원되지 않을 수 있다.

glance를 통한 Thread별 CPU사용량

HP의 경우는 glance라는 모니터링 툴을 통해 쓰레드 별 CPU사용량을 확인할 수 있다.

Shell> glance 를 통해 실행

<s> 키를 누르면 특정 process id를 입력할 수 있다.

<G> 키를 누르면 해당 프로세스의 쓰레드 별 CPU 사용량을 확인할 수 있다.

pstack

SUN의 pstack과 동일한 결과를 보여 준다.

Shell> pstack <process id>

---------- **lwpid : 3486042** ----------

0: c000000000446910 : (unknown) () (unknown)

1: c0000000001a75a0 : (unknown) () (unknown)

2: c0000000000e1130 : (unknown) () (unknown)

3: c0000000000e40c0 : (unknown) () (unknown)

4: 4000000001330fd0 : rpxSender::sleepForNextConnect()() + 0x3b0
(/home/ckh0618/altibase_home/bin/altibase)

5: 4000000001340cc0 : rpxSender::attemptHandshake(idBool*)() + 0x4c0
(/home/ckh0618/altibase_home/bin/altibase)

6: 4000000001326f80 : rpxSender::run()() + 0x1a0 (/home/ckh0618/altibase_home/bin/altibase)

7: 4000000001e108a0 : idtBaseThread::staticRunner(void*)() + 0x60
(/home/ckh0618/altibase_home/bin/altibase)

8: c0000000000fa220 : (unknown) () (unknown)

lwpid가 쓰레드의 고유번호를 의미한다. 동일하게 lwpid로 단락을 구분하고 아래에서 위로 해석한다. 위의 경우는 이중화Sender Thread가 상대편과 연결하기 위한 동작을 보여 주고 있다.

HP의 경우 pstack이 없다면 gdb를 통해 획득할 수 있다. 다만, 주의할 것은 gdb버전이 매우 낮은 경우 쓰레드 정보를 보려고 할 경우 해당 프로세스가 비정상 종료할 수 있음으로 고객에게 사전에 위험성을 알리고 작업하도록 한다.

Shell> gdb $ALTIBASE_HOME/bin/altibase <process id> (gdb를 실행)

(gdb) thread apply all bt (gdb 실행된 상태에서 수행)

Thread 339 (system thread 3486042):

#0 0xc000000000446910:0 in __ksleep+0x30 () from /usr/lib/hpux64/libc.so.1

#1 0xc0000000001a75a0:0 in __mxn_sleep+0x1080 () from /usr/lib/hpux64/libpthread.so.1

#2 0xc0000000000e1130:0 in <unknown_procedure> + 0x1210 ()from /usr/lib/hpux64/libpthread.so.1

#3 0xc0000000000e40c0:0 in pthread_cond_timedwait+0x160 ()from /usr/lib/hpux64/libpthread.so.1

#4 0x4000000001330fd0:0 in rpxSender::sleepForNextConnect ()

at /home/mycomman/work/altidev4/src/rp/rpx/rpxSender.cpp:1333

#5 0x4000000001340cc0:0 in rpxSender::attemptHandshake ()

at /home/mycomman/work/altidev4/src/rp/rpx/rpxSenderHandshake.cpp:89

#6 0x4000000001326f80:0 in rpxSender::run ()

at /home/mycomman/work/altidev4/src/rp/rpx/rpxSender.cpp:750

#7 0x4000000001e108a0:0 in idtBaseThread::staticRunner ()

at /home/mycomman/work/altidev4/src/id/idt/idtBaseThread.cpp:104

#8 0xc0000000000fa220:0 in __pthread_bound_body+0x190 ()

from /usr/lib/hpux64/libpthread.so.1

다른 운영 체제에서도 gdb/dbx등의 사용은 동일하다.

pfiles

SUN의 pfiles와 동일한 결과를 보여 준다.

Shell> pfiles <process id>

0: S_ISREG mode:666 dev:64,65537 ino:8490324 uid:124 gid:20 size:530024 flags = O_WRONLY|O_APPEND|O_LARGEFILE file = /home/ckh0618/altibase_home/trc/altibase_boot.log

1: S_ISREG mode:666 dev:64,65537 ino:8490373 uid:124 gid:20 size:8466361 flags = O_WRONLY|O_APPEND|O_LARGEFILE file = /home/ckh0618/altibase_home/trc/altibase_sm.log

시스템 로그

HP에서 시스템 로그를 확인하기 위해 다음과 같이 확인한다.

Shell> vi /var/adm/syslog/syslog.log

Feb 24 10:32:07 rx5670 vmunix: System Console is on the Built-In Serial Interface

Feb 24 10:32:07 rx5670 vmunix: igelan0: INITIALIZING HP A6794-60001 PCI 1000Base-T at hardware path 0/1/1/0/4/0

Feb 24 10:32:07 rx5670 vmunix: Logical volume 64, 0x3 configured as ROOT

Feb 24 10:32:07 rx5670 vmunix: Logical volume 64, 0x2 configured as SWAP

Feb 24 10:32:07 rx5670 vmunix: Logical volume 64, 0x2 configured as DUMP

Linux

Thread별 CPU사용량

리눅스(Linux)에서 쓰레드 별 CPU를 top명령어로도 확인이 가능하다. Top –H 옵션으로 수행하면 쓰레드 별로 조회가 된다. 하지만 설치된 procps version이 3.2.7 이상이어야 가능하다.

아닌 경우 간단하게 다음과 같이 확인한다. (역시 낮은 버전에서는 잘 지원 안됨)

Shell> ps -LFm –p <process id>

UID	PID	PPID	**LWP**	**C**	NLWP	SZ	RSS	PSR	STIME	TTY	TIME	CMD
lim272	6870	1	-	0	19	204217	355068	-	19:03	?	00:00:01	altibase -p boot from
lim272	-	-	6870	0	19	-	-	7	19:03	-	00:00:00	-
lim272	-	-	6871	0	19	-	-	0	19:03	-	00:00:00	-
lim272	-	-	6872	0	19	-	-	6	19:03	-	00:00:00	-

LWP가 쓰레드 별 고유번호이며 C항목에 출력 되는 값이 CPU사용율이다.

pstack

SUN과 동일한 결과를 보여 준다. 다만, 일부 커널이 낮은 버전의 경우는 pstack이 지정된 쓰레드의 스택만 보여 주는 경우도 있다.

```
Shell> pstack <process id>
Thread 3 (Thread 1335986528 (LWP 7171)):
#0  0x000000331c8bf0a6 in __select_nocancel () from /lib64/tls/libc.so.6
#1  0x0000000000a05118 in cmnDispatcherSelectSOCK ()
#2  0x00000000009faa21 in cmiSelectDispatcher ()
#3  0x000000000063ef58 in rpcExecutor::run ()
#4  0x0000000000a28c31 in idtBaseThread::staticRunner ()
#5  0x000000331d30610a in start_thread () from /lib64/tls/libpthread.so.0
#6  0x000000331c8c6003 in clone () from /lib64/tls/libc.so.6
#7  0x0000000000000000 in ?? ()
Thread 2 (Thread 1315006816 (LWP 7172)):
```

분석하는 방법은 다른 운영체제의 결과 보는 방법과 동일하다. gdb가 설치된 경우는 gdb를 이용해 HP에서 설명한 바와 같이 결과를 얻어낼 수도 있다.

사용 중인 파일 목록 확인

리눅스에서 lsof와 같은 별도의 유틸을 설치하지 않는다면 다음과 같이 한다.

```
Shell> ls –l /proc/<process id>/fd
lrwx------  1 lim272 lim272 64 Feb 26 19:34 20 -> /home3/altibase_home/logs/logfile49
lrwx------  1 lim272 lim272 64 Feb 26 19:34 21 -> /home3/altibase_home/logs/logfile50
lrwx------  1 lim272 lim272 64 Feb 26 19:34 22 -> /home3/altibase_home/logs/logfile51
lrwx------  1 lim272 lim272 64 Feb 26 19:34 23 -> /home3/altibase_home/logs/logfile52
lrwx------  1 lim272 lim272 64 Feb 26 19:34 24 -> /home3/altibase_home/logs/logfile53
lrwx------  1 lim272 lim272 64 Feb 26 19:34 25 -> socket:[16080]
```

리눅스는 /proc/<process id> 경로 아래로 각종 정보를 확인할 수 있음으로 관련된 사항을 참고하도록 한다.

시스템 로그

/var/log/에 존재하는 파일을 확인한다. 일반적으로 messages파일을 확인하도록 한다.

알티베이스㈜
서울특별시 구로구 구로 3 동 182-13
대륭포스트 2 차 1008 호
02-2082-1000
http://www.altibase.com

대전사무소
대전광역시 서구 둔산동 921
주은리더스텔 901 호
042-489-0330

기술지원본부
서울특별시 구로구 구로 3 동 182-13
대륭포스트 2 차 908 호
02-2082-1000

솔루션센터
02-2082-1114
http://support.altibase.com