

# Futásidő, memóriahasználat monitorozása (profiling)

Majzik István és Micskei Zoltán  
Budapesti Műszaki és Gazdaságtudományi Egyetem  
Méréstechnika és Információs Rendszerek Tanszék  
<http://www.inf.mit.bme.hu/>

## Tartalomjegyzék

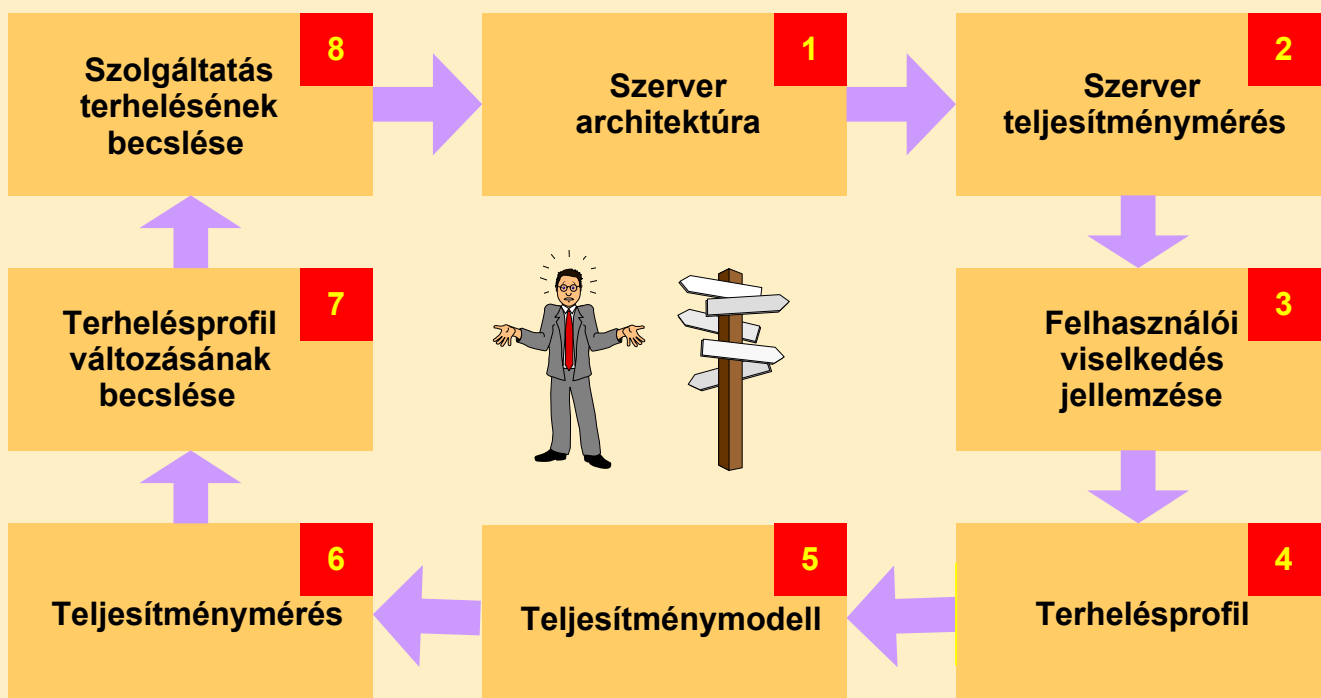
- Motiváció
  - Célkitűzés: Belső működés analízise
- A profiling általános problémái
  - Felműszerezés, triggerelés, regisztrálás
  - Szoftver és hardver monitorozás
- Futásidő profiling
  - Teljesítményproblémák felderítése
- Memóriahasználat profiling
  - Memóriaproblémák proaktív monitorozása

# Célkitűzések

- Mi a profiling?
  - Dinamikus program analízis (belső működés vizsgálata),
  - a végrehajtás (tesztelés) során gyűjtött információk alapján,
  - a hibák, optimalizálási lehetőségek felderítése érdekében
- Az információgyűjtés (monitorozás)
  - Futásidő adatok felvétele (pl. hívásidő, hívási gyakoriság)
  - Memóriafooglalás követése (pl. elmaradt felszabadítás)
- A profiling analízis része: A gyűjtött információ felhasználása
  - Hibakeresés
    - Hol állt le a program?
  - Teljesítmény viszonyok vizsgálata
    - Hol vannak szűk keresztmetszetek? Mit érdemes gyorsítani?
  - Memóriahasználati viszonyok
    - Mit lehet optimalizálni?
- Amivel nem foglalkozunk:
  - Teljesítmény modellezés, kapacitástervezés, változásbecslés, optimalizálás, benchmark tervezés, felhasználói profilok felvétele, ...

3

## Teljesítményoptimalizálás: Külön tudomány 😊



4

# Tartalomjegyzék

- Motiváció
  - Célkitűzés: Belső működés analízise
- A profiling általános problémái
  - Felműszerezés, triggerelés, regisztrálás
  - Szoftver és hardver monitorozás
- Futásidő profiling
  - Teljesítményproblémák felderítése
- Memóriahasználát profiling
  - Memóriaproblémák proaktív monitorozása

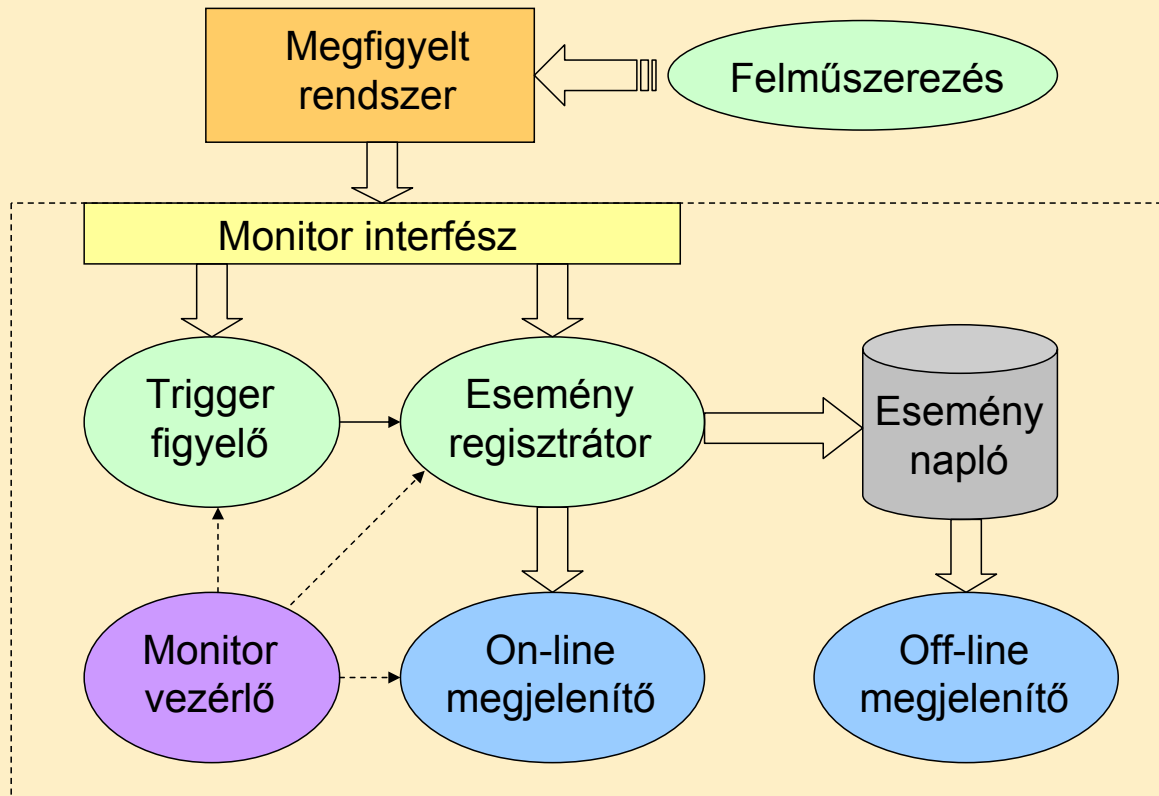
5

## A monitorozás alpműveletei

- Információgyűjtés:
  - Időadatok, erőforráshasználat, kommunikáció, ...
- Feladatok:
  - Információ hozzáférés: **Felműszerezés**
    - Szoftver úton: Extra utasítások beszúrása
      - Forráskód alapján, bináris kódban, vagy futási időben
    - Hardver úton: Csatlakozás a rendszerbuszra, csatornára, ...
  - Információ szűrés: **Triggerelés**
    - Szoftver úton: Extra utasítások megoldják
    - Hardver úton: Jelek figyelése (pl. buszon)
  - Információ tárolás: **Regisztrálás**
    - Szoftver úton: Naplózás szoftver modulból
    - Hardver úton: Jelek mintavételezése („logikai analizátor”)

6

# Általános monitorozási séma



7

## Példa: Előfeldolgozás (felműszerezés)

```
for (i=0; i<9; i++) {  
  V(sm1);  
  if (i==a) {  
    P(sm1);  
  } else {  
    P(sm2);  
  }  
}
```

Elő-  
feldolgozó

```
for (i=0; i<9; i++) {  
  EV(V1); V(sm1);  
  if (i==a) {  
    EV(P1); P(sm1);  
  } else {  
    EV(P2); P(sm2);  
  }  
}
```

```
void switch_up() {  
  if( gear == 5 ) {  
    error();  
    return;  
  }  
  set_gear( gear+1 );  
}
```

```
char _TABLE[NUM_BLOCK / 8];  
void switch_up() {  
  if( gear == 5 ) {  
    error();  
    _TABLE[17/8] |= (1<<(17%8));  
    return;  
  }  
  set_gear( gear+1 );  
  _TABLE[18/8] |= (1<<(18%8));  
}
```

## A profiling (monitorozás) problémái

- **Beavatkozás:** Ha a rendszer erőforrásait használjuk, akkor megváltoztatjuk a rendszer viselkedését
  - Példa: Időzítések, eseménysorrend eltérő lesz
  - Megoldás: Hardver monitorozás, korrekció, bennhagyás
- **Szemantikai hézag:** A megfigyelt információból a szükséges információ származtatása
  - Példa: Processzor buszjelekből erőforráshasználatra következtetni
  - Megoldás: Rugalmas (szoftver) triggerelés vagy off-line analízis
- **Globális jellemzők származtatása:** Elosztott rendszerek esetén a lokális információból globális jellemzők
  - Példa: Körkörös várakozás detektálása
  - Megoldás: Központi monitor, lokális monitorok szinkronizálása

9

## Általános megoldások áttekintése

Monitorozás típusa / feladat	Felműszerezés	Triggerelés	Regisztrálás
Szoftver (rugalmas, beavatkozó)	Trigger utasítások (aut.) beszúrása	Trigger utasítások közvetlenül	Alkalmazás, vagy monitor processz
Hardver (nem beavatkozó, de specifikus)	Közvetlen csatlakozás	Hardver komparátor	Lokális vagy távoli tároló
Hibrid (rugalmas, kevésbé beavatkozó)	Trigger utasítások (aut.) beszúrása	Trigger utasítások közvetlenül	Hardver (lokális vagy távoli tároló)

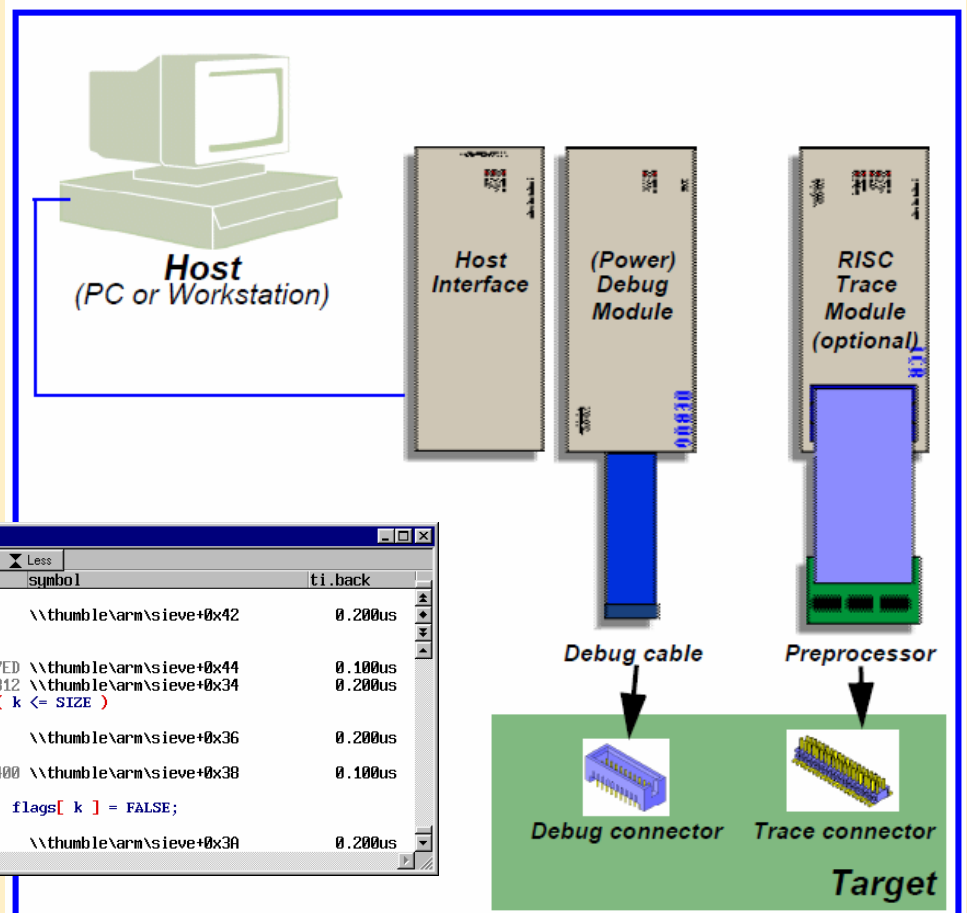
10

# Példa: Hardver monitorozás

Lauterbach

TRACE32

- Bus trace, program trace
- 512 Kframes trace memory
- 94 channels
- 36 bit time stamp  
25 ns resolution



record	run	address	cycle	d.l	symbol	ti.back
-00000023	f	add r3,r3,r7		3101	\\thumb\varm\sieve+0x42	0.200us
696		T:00001BBE	fetch			
-00000022	f	b 0x1BB0		E7ED	\\thumb\varm\sieve+0x44	0.100us
-00000021	f	T:00001BC0	fetch	2B12	\\thumb\varm\sieve+0x34	0.200us
692		T:00001BB0	fetch		while ( k <= SIZE )	
-00000020	f	cmp r3,#0x12		DC04	\\thumb\varm\sieve+0x36	0.200us
		T:00001BB2	fetch			
-00000019	f	bgt 0x1BBE		2400	\\thumb\varm\sieve+0x38	0.100us
693		T:00001BB4	fetch		{	
694					flags[ k ] = FALSE;	
-00000018	f	mov r4,#0x0		4005	\\thumb\varm\sieve+0x3A	0.200us
		T:00001BB6	fetch			

## Példa: Java Virtual Machine Tool Interface (JVMTI)

- Java Platform Debugger Architecture része
  - Korábbi Java Virtual Machine Profiler Interface (JVMPPI) és a Java Virtual Machine Debug Interface (JVMDI) helyett
- Natív (C, C++) programozási interfész
  - Futtatott alkalmazások állapotának vizsgálata (profiling, debugging)
  - Alkalmazás végrehajtás vezérlése JVM-en
- Kétirányú interfész
  - Események kliens (agent) programok számára (callback fv.)
    - Pl. ThreadStart, ThreadEnd, ClassLoad, FieldAccess, MethodEntry,...
  - Függvények biztosítása vezérléshez, lekérdezéshez
    - Pl. SuspendThread, StopThread, GetThreadInfo, GetStackTrace,...
- Java bytecode felműszerezése lehetséges a kliens által (pl. metódusszintű monitorozáshoz)
  - Statikus (előzetesen)
  - Betöltési időben (load-time): ClassFileLoadHook event
  - Futási időben (run-time): RetransformClasses function

## Példa: Szoftver monitorozás AOP felhasználásával

- Monitorozás: Teljes szoftvert „átszövő” beavatkozás
  - Sok helyen kell beavatkozni a regisztráláshoz
- Beavatkozások modularizálása aspektusokban
  - Megadható a beavatkozás helye (reguláris kifejezéssel)
    - Vágási pont (pointcut): hová kell kapcsolódni?
    - Kapcsolódási pont (join point): aspektus interakció helye
      - pl. `send*( )` - minden `send` kezdetű metódus esetén
  - Újrahasználható a beavatkozás
    - Beavatkozás (advice): mit kell ott csinálni?
      - Pl. naplófájlba írás a `send*( )` végrehajtásáról
- Fordítás során:
  - Felműszerezés: Aspektus és eredeti kód összefésülése

13

## Példa: Aspektus-orientált minta

Eredeti forráskód:

```
class Controller {  
    public int send(int msg) {  
        <<sending message>>  
    }  
}
```

A `send()` hívást szeretnénk regisztrálni

Aspektus kód:

```
public aspect SimpleLog {  
    pointcut loggedCall(int msg):  
        execution(public int Controller.send*(..)) && args(msg);  
  
    before(int input): loggedCall(msg) {  
        <<log request(msg)>>  
    }  
  
    after() throwing exception (Exception e): loggedCall() {  
        <<log exception>>  
    }  
}
```

Vágási pont definiálás `loggedCall` névvel

Vágási pont (trigger):  
`send()` végrehajtása

Vágási pont előtt avatkozunk be

Beavatkozás (regisztrálás)

Kivétel dobásának regisztrálása

15

# Tartalomjegyzék

- Motiváció
  - Célkitűzés: Belső működés analízise
- A profiling általános problémái
  - Felműszerezés, triggerelés, regisztrálás
  - Szoftver és hardver monitorozás
- **Futásidő profiling**
  - Teljesítményproblémák felderítése
- Memóriahasználat profiling
  - Memóriaproblémák proaktív monitorozása

16

## Futásidő profilerek

- **Alapfeladatok:**
  - Függvényhívási hierarchia felderítése, hívási gyakoriság hozzárendelése
  - Az egyes függvényekben **eltöltött idő** hozzárendelése
  - Annotált forrás: utasítások végrehajtási gyakorisága
  - Célkörnyezetben futtatva használhatók az eredmények
- **Egyszerű példa: gprof**
  - Program felműszerezése: speciális fordítással  
cc -pg prog.c -o prog (opció; gcrt0.o, libc\_p.a)
  - Futás közbeni adatgyűjtés:  
gmon.out adatfájlba
  - Off-line kiértékelés:  
gprof prog gmon.out

17



## Példa: gprof jelentés: Flat profile

% time	cumulative seconds	self seconds	calls	self us/call	total us/call	name
33.65	0.71	0.71	4	177500.00	177500.00	RELERR
28.91	1.32	0.61	4	152500.00	152500.00	RELFINE
13.27	1.60	0.28	60	4666.67	4666.67	RELCOARSE
10.90	1.83	0.23	32	7187.50	7187.50	INTERPOL
6.16	1.96	0.13	4	32500.00	32500.00	RESFINE
2.84	2.02	0.06	28	2142.86	2142.86	RESCOARSE
1.90	2.06	0.04	1	40000.00	40000.00	write_res
1.42	2.09	0.03	1	30000.00	30000.00	INIT
0.95	2.11	0.02	8	2500.00	2500.00	PUTZERO
0.00	2.11	0.00	1	0.00	0.00	OUTPUT

- **Time:** Percentage of the total running time of program used by this function
- **Cumulative seconds:** Sum of the seconds accounted for by this function and those listed above it
- **Self seconds:** The number of seconds accounted for by this function alone
- **Self us/call:** The average number of microseconds spent in this function per call
- **Total us/call:** The average number of microseconds spent in this function and its descendents per call

18

## Példa: gprof jelentés: Call graph

index	% time	self	children	called	name
[1]	100.0	0.00	2.11		main [1]
		0.71	0.00	4/4	RELERR [2]
		0.61	0.00	4/4	RELFINE [3]
		0.28	0.00	60/60	RELCOARSE [4]
		0.23	0.00	32/32	INTERPOL [5]
		0.13	0.00	4/4	RESFINE [6]
		0.06	0.00	28/28	RESCOARSE [7]
		0.04	0.00	1/1	write_res [8]
		0.03	0.00	1/1	INIT [9]
		0.02	0.00	8/8	PUTZERO [10]
-----					
[2]	33.6	0.71	0.00	4	main [1]
		0.71	0.00	4	RELERR [2]
-----					
[3]	28.9	0.61	0.00	4	main [1]
		0.61	0.00	4	RELFINE [3]

- **Self:** Total amount of time spent in this function
- **Children:** Total amount of time propagated into this function by its children
- **Called:** Number of times the function was called

19

# Példa: Rational Quantify

- Automatizált feladatok:
  - Futásidő adatok gyűjtése
  - Adatok analízise és bemutatása
    - Hívási hierarchia
    - Metódusok futási ideje
- Felhasználás:
  - Problémás helyek (hívások) azonosítása: Teljesítmény szempontjából szűk keresztmetszetek
    - Melyik metódust érdemes gyorsítani, átstrukturálni?
  - Javítások hatásának ellenőrzése
    - Gyorsult-e a végrehajtás a változtatás után?

Method	Calls	Method time	M+D time	M time (% of Focus)	M+D time (% of Focus)	Avg M time	Min M time	Max M time	
JVM Garbage Collector	75	44868316,98	44868316,98	35,25	35,25	598244,23	2443,92	16613123,72	JVM int
Logger.formatDateSt...	20 0,00	16878492,69	52820771,13	13,26	41,50	843,92	749,10	12801,35	carstor
FileOutputStream.wri...	20 0,00	9844922,18	9844922,18	7,73	7,73	492,00	0,00	15543,75	java.io.l
Calendar.internalSet	2 040 034	4002867,07	4002867,07	3,14	3,14	1,96	1,64	15624,89	java.util
GregorianCalendar.ti...	120 032	3595607,17	6789004,86	2,82	5,33	29,96	25,75	3908,20	java.util
GregorianCalendar.c...	120 032	3452694,37	13651229,90	2,71	10,72	28,77	24,37	5325,30	java.util
Date.getField	120 030	1784801,13	20207988,46	1,40	15,88	14,87	11,27	8492,52	java.util
AbstractList\$Itr.next	315 740	1765584,49	2714147,85	1,39	2,13	5,59	4,71	6090,37	java.util
TestDriver.addAndR...	10	1661241,00	82032486,10	1,31	64,45	166124,10	146859,16	207916,06	carstor
Vector.add	325 757	1430512,56	2188086,08	1,12	1,72	4,39	3,26	4270,87	java.util
AbstractList\$Itr.hasN...	335 740	1400103,59	1922007,70	1,10	1,51	4,17	3,38	2771,53	java.util
SingleByteEncoder.e...	20 020	1000981,51	1706719,29	0,79	1,34	50,00	30,82	3909,54	sun.nio
Object.clone	120 030	932350,68	932350,68	0,73	0,73	7,77	0,00	14199,04	java.lar
carstorage.CarPart	100 030	870253,37	1498628,61	0,68	1,18	8,70	6,51	4986,63	carstor
ThreadLocal.get	85 126	856203,36	1755414,72	0,67	1,38	10,06	7,00	8016,96	java.lar
String.charAt	428 838	637476,97	637476,97	0,66	0,66	1,35	1,34	2509,34	java.lar
System.arraycopy	306 073	832533,97	832533,97	0,65	0,65	2,72	1,27	2035,82	java.lar
Integer.toString	120 030	789261,49	4318548,19	0,62	3,39	6,58	5,14	6109,07	java.lar
Integer.toString	120 030	762285,90	2592852,44	0,60	2,04	6,35	1,33	3749,57	java.lar
Calendar.get	120 031	757033,27	1283736,37	0,59	1,01	6,31	5,30	1050,86	java.util
StringBuffer.append	110 531	708338,34	1744617,69	0,56	1,37	6,41	3,56	2025,47	java.lar
carstorage.Wheel	40 030	686297,82	1759679,92	0,54	1,38	17,16	13,01	2524,85	carstor
Vector.ensureCapac...	325 774	667724,30	757643,71	0,52	0,60	2,05	1,54	5237,05	java.util
Calendar.setTimeInM...	120 032	665378,14	14745400,83	0,52	11,58	5,54	4,04	4520,94	java.util
TimeZone.clone	120 033	663536,45	1595839,28	0,52	1,25	5,53	4,36	2511,05	java.util
lang.Integer	120 031	633571,99	828346,39	0,50	0,65	5,28	3,43	4481,47	java.lar
CarPart.setPrice	100 030	628375,24	628375,24	0,49	0,49	6,28	4,40	4770,70	carstor
String.getChars	130 615	586556,51	894593,90	0,46	0,70	4,49	3,14	5033,81	java.lar
carstorage.Car	10 030	580551,83	2989685,29	0,46	2,35	58,05	51,21	5085,30	carstor
lang.Character	315 740	575491,95	575491,95	0,45	0,45	1,82	1,23	4212,11	java.lar
ZoneInfo.getOffsets	120 032	565142,19	883072,56	0,44	0,69	4,71	3,38	10095,51	sun.util
String.valueOf	120 030	558132,59	3529286,70	0,44	2,77	4,65	3,76	1137,56	java.lar
Vector.size	335 820	522084,69	522084,69	0,41	0,41	1,55	1,27	6052,17	java.util
Logger.carAckled	10 030	519821,82	28141281,44	0,41	22,11	51,98	40,96	9771,66	carstor
StringBuffer.append	317 331	517379,63	517750,95	0,41	0,41	1,63	1,32	3815,06	java.lar
GregorianCalendar.m...	120 032	504407,68	762172,62	0,40	0,60	4,20	1,60	2155,54	java.util
GregorianCalendar.f...	240 034	503738,33	503738,33	0,40	0,40	2,10	1,48	537,36	java.util

## A teljesítményproblémák tipikus okai

- **Haszontalan számítás**
  - Nem törölt kódrészletek, amik erőforrást foglalnak
  - Igény nélküli, „alapértelmezett” szolgáltatások
- **Felesleges újraszámítás**
  - Részeredmények újraszámítása tárolás helyett
- **Rendszerszolgáltatások mértéktelen igénybevétele**
  - Lassú környezetváltás
- **Foglalt erőforrásokra való várakozás**
  - Szinkronizáció másik folyamattal
  - Kommunikációs csatorna foglaltsága
  - Állományok foglaltsága

22

## A teljesítményproblémák megoldási lehetőségei

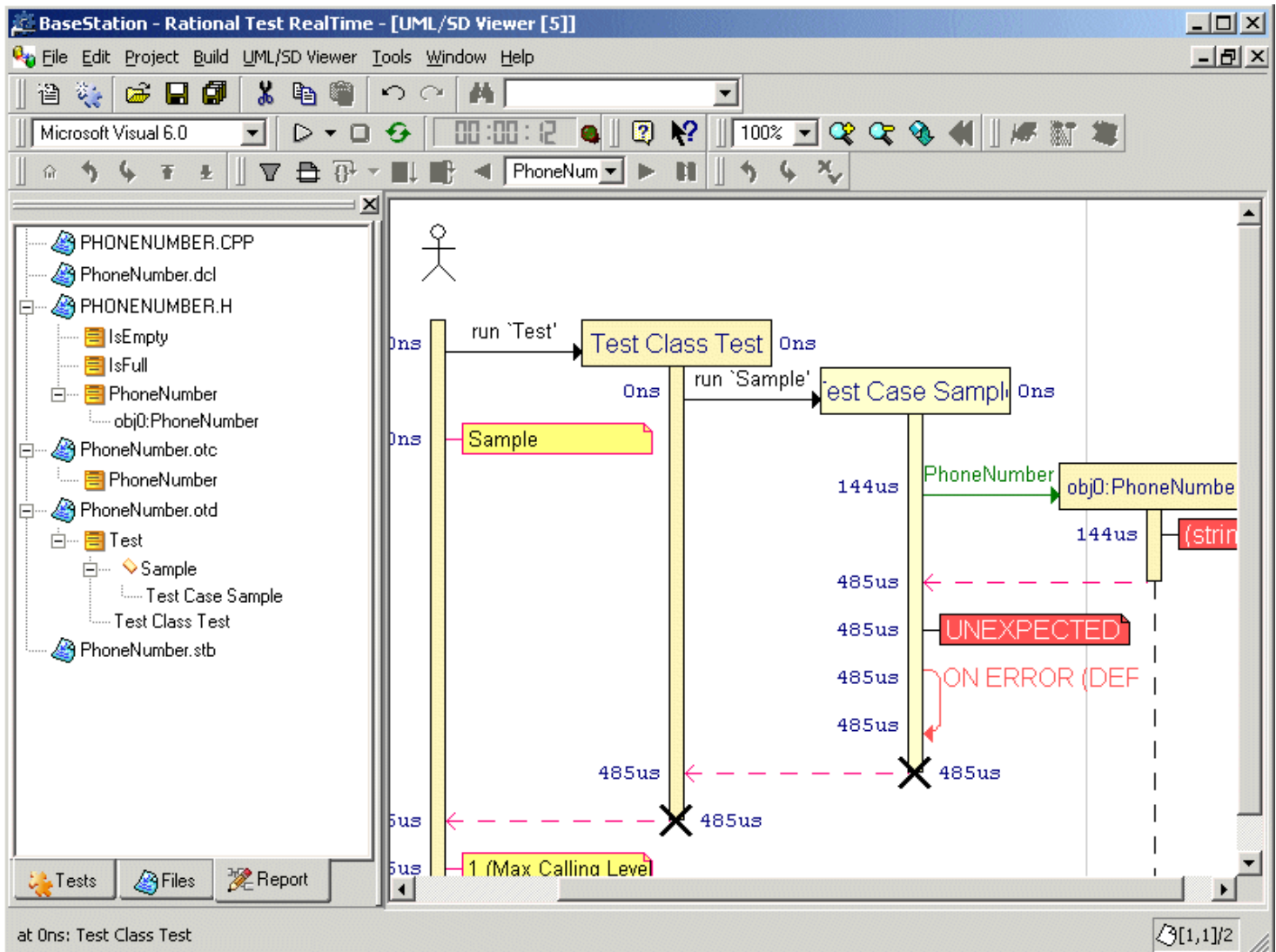
- **Program átstrukturálás**
  - Felesleges számítások, hívások törlése
  - Többszálú működés kialakítása (többmagos rendszerekre)
- **Optimalizált számítások**
  - Ismétlés elkerülése ideiglenes tárolással a programon belül
- **Halasztott számítások**
  - Felhasználó szempontjából nem kritikus időben végezve
- **Kevesebb rendszerhívás**
  - Adatok időleges tárolása (cache)
  - Foglalt erőforrások státusz figyelése
- **Hatékonyabb algoritmusok**
  - Rendezés, keresés, ... algoritmusok: Sokféle verzió
  - Speciális utasításkészlet használata: Párhuzamosítás
  - Koprocesszor (GPU) használata egyes részfeladatokhoz

23

## Példa: Rational Test RealTime

- Integrációs tesztelés támogatása
- Scenariók felvétele
  - Script rögzítés
- Együttműködés megjelenítése
  - Szekvencia diagram formájában
- Teljesítmény jellemzők megjelenítése
  - Időadatok rögzítése

24



BaseStation - Rational Test RealTime - [D:\W2KPRO~1\Rational\TESTRE~1\examples\BaseStation\tests\MobilePhoneVT.pts]

File Edit Project Build Editor Tools Window Help

Microsoft Visual 6.0

00:00:00 200%

BaseStation

- BaseStation\_Rational
  - Introduction
  - Interactive
  - UmtsCode
  - PhoneNumber
  - MobilePhoneVT
    - MobilePhoneVT.pts

**TERMINATION**

```
CALL tcpsock_close(sock)
CALL tcpsock_end()

END TERMINATION

SCENARIO main

SCENARIO connect

INSTANCE phone1:

COMMENT Send connect request to the Base Station
COMMENT And wait for reply

VAR mConnect, INIT={command=>cmd_connection_request,simCardId=>20}
SEND (mConnect,BaseStation)

DEF MESSAGE mResponse, EV={command=>cmd_connection_established}
WAITTIL ( MATCHING(mResponse,BaseStation), WTIME>1000 )

END INSTANCE

END SCENARIO
```

Tests Files Report

Ready Line: 42 Col: 15

BaseStation - Rational Test RealTime - [UML/SD Viewer [2]]

File Edit Project Build UML/SD Viewer Tools Window Help

Microsoft Visual 6.0

00:00:16 75%

MobilePhor

MobilePhoneVT.xrd

- SUT
  - BaseStation
    - SUT BaseStation
  - VT
    - phone1\_0
      - VT phone1\_0

```
sequenceDiagram
    participant VT as VT phone1_0
    participant SUT as SUT BaseStation
    Note over VT: 0ns
    VT->>SUT: Initialization
    Note over SUT: 25ms
    SUT->>VT: main
    Note over VT: 43ms
    SUT->>VT: connect
    Note over VT: 43ms
    VT->>SUT: mConnect
    Note over SUT: 50ms
    SUT->>VT: WAITTIL
    Note over VT: 52ms
    SUT->>VT: MATCHING(mResponse)
    Note over VT: 52ms
    SUT->>VT: MATCHING(mResponse)
    Note over SUT: 4s 40ms
    SUT->>VT: END WAITTIL
    Note over VT: 4s 42ms
    SUT->>VT: END connect
    Note over VT: 4s 42ms
    SUT->>VT: call_busy
    Note over VT: 4s 43ms
    VT->>SUT: mCall
    Note over SUT: 4s 44ms
    SUT->>VT: WAITTIL
    Note over VT: 4s 45ms
    SUT->>VT: MATCHING(mResponse)
    Note over VT: 4s 45ms
    SUT->>VT: MATCHING(mResponse)
    Note over SUT: 8s 46ms
    SUT->>VT: END WAITTIL
    Note over VT: 8s 48ms
```

Tests Files Report

Ready

## Java profiler eszközök

- Rengeteg eszköz:
  - <http://java-source.net/open-source/profilers>
- jvisualvm
  - JDK belépített eszközöket használ
- YourKit Java Profiler
- Quest JProbe
- JIP – Java Interactive Profiler
- Netbeans Profiler
- ...
- Eclipse:
  - [Test & Performance Tools Platform](#) (TPTP Id. később)

28

## Tartalomjegyzék

- Motiváció
  - Célkitűzés: Belső működés analízise
- A profiling általános problémái
  - Felműszerezés, triggerelés, regisztrálás
  - Szoftver és hardver monitorozás
- Futásidő profiling
  - Teljesítményproblémák felderítése
- **Memóriahasználat profiling**
  - Memóriaproblémák proaktív monitorozása

29



## Jellegzetes memóriahibák

- (Statikusan detektálható hibák
  - Pl. típuskonverziós hibák)
- Memória szivárgás: **hiányzó kódrészlet** miatt
  - Lefoglalt, de fel nem szabadított memóriaterületek (C++) → elfogy a memória...
  - Szükségtelen hivatkozások miatt foglalt objektumok (Java) → szemétyűjtés nem működik...
- Illegális memóriahasználát: **hibás kód** miatt
  - Tömbök határainak túlcímzése
  - Inicializálatlan memória olvasása
  - Írás felszabadított memóriaterületre
  - Allokálatlan memóriához való hozzáférés

30

## Nehézségek a tesztelésben

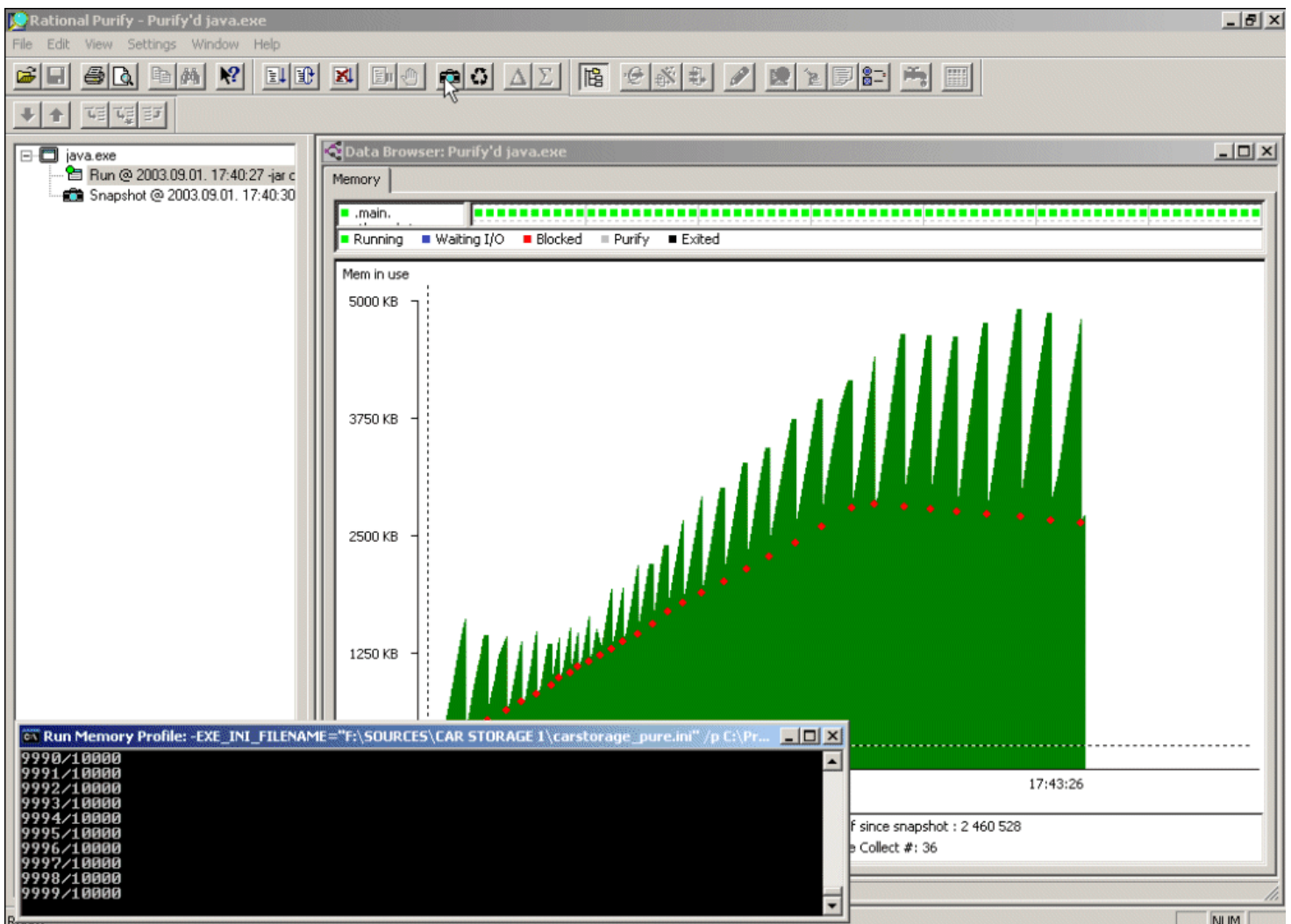
- „Rejtélyes” hatások
  - Hatások csak későn és más formában jelentkeznek (pl. adathiba, lassulás, fagyás, ...)
  - Hatások csak nehezen reprodukálhatók
- Ötlet: Nem a késői hatás alapján detektálni
  - Tömbök határai előtt és után „tilos zóna” beillesztése
  - Memória státuszának nyilvántartása (allokált, inicializált stb.)
  - Hivatkozások nyilvántartása és ellenőrzése (C++)
  - Allokált memória kijelzése és összehasonlítása (Java); a folyamatosan foglalt metódusok azonosítása

31

# Példa eszköz: Rational Purify

- Automatizált feladatok:
  - Memóriafooglalás rögzítése
  - Memóriakezelési hibák futásidejű detektálása
- Kiindulási alap:
  - Memóriafooglalás és -felszabadítás regisztrálása
  - Automatikusan illeszt be ellenőrző kódrészleteket (akár a forráskód ismerete nélkül)
    - Solaris: object code alapján (linkelés fázisban)
    - Windows: futtatható kód alapján
    - Java: virtuális gép profiling

32





# Példa: Memory Analyzer (Eclipse)

- Heap dump fájl vizsgálata
- Különálló RCP alkalmazás, vagy Eclipse Memory Analysis nézet

The screenshot displays the Eclipse Memory Analyzer interface. The main window shows the analysis of a heap dump file named `java_pid20049.hprof`. The Overview pane provides a summary: **Size: 63 MB**, **Classes: 12.1k**, **Objects: 1.4m**, and **Class Loader: 177**. A link to [Unreachable Objects Histogram](#) is visible. Below this, a pie chart titled "Biggest Objects by Retained Size" shows the distribution of memory usage. The largest object is 45 MB, followed by 7.6 MB, 6.1 MB, and 4.3 MB. The total size is 63 MB.

The Inspector pane on the left shows the object's profile and attributes. The attributes table is as follows:

Type	Name	Value
ref	surrogateP1	null
long	timestamp	1288598661620
boolean	changed	false
ref	iuPropertie	java.util.HashMap @ 0x74...
ref	ius	org.eclipse.equinox.intern...
ref	storage	org.eclipse.equinox.intern...
ref	subProfileIc	null
ref	translation:	null
ref	capabilityIc	null
ref	propertiesIc	null
ref	idIndex	null
ref	parentProfil	null
ref	profileId	org.eclipse.equinox.intern...