

Profiling mit gprof

André Preußner
Testen von Software
SS 2004

Was ist Profiling ?

- Profiling zeigt,
 - wo das Programm seine Zeit verbringt
 - welche Funktionen aufgerufen werden
- Profiling hilft beim Auffinden,
 - von zu langsamen Funktionen
 - von zu häufigen / seltenen Funktionsaufrufen

Wie funktioniert Profiling ?

- Der Compiler verändert die Funktionen, so dass sie die Information speichern, von wem sie aufgerufen wurde und wie oft
- Während der Laufzeit wird ein Histogramm mit der aktuellen Position des Programmzählers geführt
- Ein Clock-Handler macht ca. alle 100 Schritte Einträge in das Histogramm
- Spezielle mit '-pg' kompilierte C-Bibliotheken werden in das Programm eingebunden

Wie funktioniert Profiling ? (2)

- Betrachtet wird nur die tatsächliche Laufzeit des Programmes; evtl. Systemaktivitäten wie Swapping werden nicht erfasst
 - Nachteil: Funktionen, die aufgrund grosser Datenmengen ein Swapping verursachen, scheinen schneller zu laufen
 - Vorteil: Zeit, die von anderen Programmen verbraucht wurde, verfälscht nicht das Ergebnis

Wie profile ich ein Programm ?

- Kompilieren und Linken des Programms
- Programm ausführen um Profil-Daten zu erstellen
- gprof ausführen und Profil-Daten analysieren

Kompilieren und Linken

- Kompilieren und Linken mit der Option '-pg'
 - `cc -g -c myprog.c utils.c -pg`
- Linken mit '-lp_c' linkt eine mit '-pg' kompilierte Version der Standardbibliothek
- Nur für mit '-pg' kompilierte Module werden Profiling-Daten gesammelt

Ausführen des Programms

- Programm normal ausführen
- Während der Ausführung werden automatisch Profiling-Daten gesammelt
- Nur für ausgeführte Funktionen werden Daten erstellt
- Daten werden bei Beendigung des Programmes in der Datei 'gmon.out' abgelegt
- Programm muss normal terminieren

Interpretation der Profiling-Daten

- Ausgabe von gprof enthält
 - Das flache Profil
 - Den Aufrufgraphen

Das “Flache Profil”

- Zeigt die Laufzeit aller relevanten Funktionen

```
Flat profile:
```

```
Each sample counts as 0.01 seconds.
```

% time	cumulative seconds	self seconds	calls	self ms/call	total ms/call	name
33.34	0.02	0.02	7208	0.00	0.00	open
16.67	0.03	0.01	244	0.04	0.12	offtime
16.67	0.04	0.01	8	1.25	1.25	memccpy
16.67	0.05	0.01	7	1.43	1.43	write
16.67	0.06	0.01				mcount
0.00	0.06	0.00	236	0.00	0.00	tzset
0.00	0.06	0.00	192	0.00	0.00	tolower
0.00	0.06	0.00	47	0.00	0.00	strlen
0.00	0.06	0.00	45	0.00	0.00	strchr
0.00	0.06	0.00	1	0.00	50.00	main
0.00	0.06	0.00	1	0.00	0.00	memcpy
0.00	0.06	0.00	1	0.00	10.11	print
0.00	0.06	0.00	1	0.00	0.00	profil
0.00	0.06	0.00	1	0.00	50.00	report

```
...
```

Der Aufruf-Graph

- Zeigt die Laufzeit aller relevanten Funktionen und der von ihnen aufgerufenen Funktionen
- Hilft, Funktionen zu finden, die durch den Aufruf anderer Funktionen viel Zeit verbraucht haben

Der Aufruf-Graph (2)

granularity: each sample hit covers 2 byte(s) for 20.00% of 0.05 seconds

index	% time	self	children	called	name
[1]	100.0	0.00	0.05		<spontaneous> start [1]
		0.00	0.05	1/1	main [2]
		0.00	0.00	1/2	on_exit [28]
		0.00	0.00	1/1	exit [59]

[2]	100.0	0.00	0.05	1/1	start [1]
		0.00	0.05	1	main [2]
		0.00	0.05	1/1	report [3]

[3]	100.0	0.00	0.05	1/1	main [2]
		0.00	0.05	1	report [3]
		0.00	0.03	8/8	timelocal [6]
		0.00	0.01	1/1	print [9]
		0.00	0.01	9/9	fgets [12]
		0.00	0.00	12/34	strncmp <cycle 1> [40]
		0.00	0.00	8/8	lookup [20]
		0.00	0.00	1/1	fopen [21]
		0.00	0.00	8/8	chewtime [24]
		0.00	0.00	8/16	skipspace [44]

[4]	59.8	0.01	0.02	8+472	<cycle 2 as a whole> [4]
		0.01	0.02	244+260	offtime <cycle 2> [7]
		0.00	0.00	236+1	tzset <cycle 2> [26]

Der Aufruf-Graph (3)

- Zyklen durch sich gegenseitig aufrufende Funktionen

index	% time	self	children	called	name
[3]	91.71	1.77	0	1/1	main [2]
		1.77	0	1+5	<cycle 1 as a whole> [3]
		1.02	0	3	b <cycle 1> [4]
		0.75	0	2	a <cycle 1> [5]
[4]	52.85	1.02	0	3	a <cycle 1> [5]
		0	0	0	b <cycle 1> [4]
		0	0	2	a <cycle 1> [5]
		0	0	3/6	c [6]
[5]	38.86	1.77	0	1/1	main [2]
		0.75	0	2	b <cycle 1> [4]
		0	0	1	a <cycle 1> [5]
		0	0	3	b <cycle 1> [4]
		0	0	3/6	c [6]

Optionen von gprof

- `-e Funktionsname` - im Aufrufgraphen werden keine Informationen über die angegebene Funktion aufgeführt
- `-f Funktionsname` - beschränkt den Aufrufgraphen auf die angegebene Funktion
- `-z` - zeigt alle Funktionen, auch solche, die nicht aufgerufen wurden
- `-s` - legt eine Zusammenfassung der Profiling-Daten in der Datei 'gmon.sum' ab

Statistische Ungenauigkeiten

- Profiling-Daten werden mit konstanter Meßrate erhoben --> statistische Ungenauigkeiten möglich
- Hohe Wahrscheinlichkeit bei kurzen Funktionen, gar nicht oder zweimal aufzutauchen
- Eine Messung ist umso korrekter, je grösser die gemessene Zeit gegenüber der Samplingrate ist
- Lösungen:
 - Programm länger laufen lassen
 - Mehrere Programmläufe zusammenfassen