

Depuración

- ¿Qué es un depurador?
 - ✓ Programa
 - ✓ Utilidad: encontrar/corregir errores de ejecución del programa.
- Opción de compilación para generar información para el depurador.
- Puntos de ruptura (breakpoints)
 - ✓ Marcar los puntos de interés del código fuente donde se desea parar la ejecución.
 - ✓ Permite interrumpir la ejecución del programa y visualizar información sobre él.



Depuración con kdevelop

- Kdevelop ejecuta gdb como depurador, se puede seleccionar otro programa.
- Para depurar el programa el menú Debugger proporciona las herramientas.
- Run: Ejecuta (o continúa la ejecución) el programa.
 - ✓ Si no hay puntos de ruptura será una ejecución normal.
- Run to Cursor: Idem hasta la posición actual del cursor en el código fuente.
- Step Over: Ejecuta una línea de código y para en la siguiente.
- Step Over Instruction: Ejecuta una instrucción máquina.



Depuración con kdevelop (2)

- **Step In:** Ejecuta una línea de código pero entrando en llamadas a métodos si es necesario.
- **Viewers:** Abre una ventana de diálogo donde inspeccionar a bajo nivel valores de la ejecución (desensamblar código, ver registros, memoria, etc...)
- **Interrupt:** Para la ejecución del programa.
- **Stop:** Para y sale del depurador.
- **Ver variables:** marcar y añadir a Watch.



Perfiles de ejecución de programas

gprof



gprof

- **Produce un perfil de ejecución de programas:**
 - ✓ Tiempo consumido en cada función y número de veces que ha sido llamada.
 - ✓ Para cada función $f()$:
 - Qué funciones llamaron a $f()$ y cuantas veces.
 - A qué funciones llamó $f()$ y cuantas veces.
- **Esta información permite realizar mejoras en la estructura del programa o la implementación, puesto que conocemos:**
 - ✓ Las funciones que más tiempo consumen.
 - ✓ Las funciones más invocadas.



gprof (2)

- **Para poder obtener el perfil es preciso:**
 - ✓ Compilar con la opción `-pg`
 - `g++ -pg mi_programa.cc`
 - ✓ Ejecutar el programa:
 - `./mi_programa`
 - Se genera un archivo `gmon.out` con el grafo de llamadas del programa
 - ✓ Analizar la ejecución:
 - `gprof mi_programa`



Ejemplo gprof

```
void funcion1() {
    float sum=0.0;
    int i,j;
    for (i=0; i<10000; i++)
        for (j=0; j<1000; j++)
            sum += i * j;
}

void funcion2() {
    double prod;
    int i,j;
    for (i=1; i<10000; i++)
        for (j=1; j<1000; j++)
            prod *= i * j;
}

void funcion3() {
    funcion1();
    funcion2();
    funcion1();
}
```

Código fuente

```
int main() {
    funcion1();
    funcion2();
    funcion3();
    return 0;
}
```



Ejemplo gprof (perfil, 1)

Flat profile:

Each sample counts as 0.01 seconds.

%	cumulative	self		self	total	
time	seconds	seconds	calls	ms/call	ms/call	name
89.51	7.42	7.42	2	3710.00	3710.00	funcion2
10.49	8.29	0.87	3	290.00	290.00	funcion1
0.00	8.29	0.00	1	0.00	4290.00	funcion3



Ejemplo gprof (perfil, 2)

Call graph (explanation follows)

index	% time	self	children	called	name
					<spontaneous>
[1]	100.0	0.00	8.29		main [1]
		0.00	4.29	1/1	funcion3 [3]
		3.71	0.00	1/2	funcion2 [2]
		0.29	0.00	1/3	funcion1 [4]

		3.71	0.00	1/2	funcion3 [3]
		3.71	0.00	1/2	main [1]
[2]	89.5	7.42	0.00	2	funcion2 [2]

		0.00	4.29	1/1	main [1]
[3]	51.7	0.00	4.29	1	funcion3 [3]
		3.71	0.00	1/2	funcion2 [2]
		0.58	0.00	2/3	funcion1 [4]

		0.29	0.00	1/3	main [1]
		0.58	0.00	2/3	funcion3 [3]
[4]	10.5	0.87	0.00	3	funcion1 [4]

