Herramientas de Depuración

Daniel Serpell
daniel.serpell@aplik.cl

- Por qué es necesario depurar

- Cómo depurar
  - mensajes: printf / cout.
  - “depuradores” (debuggers).
  - depuradores de memoria.
  - seguidores de llamadas (call trace).
  - análisis de ejecución (execution profile).

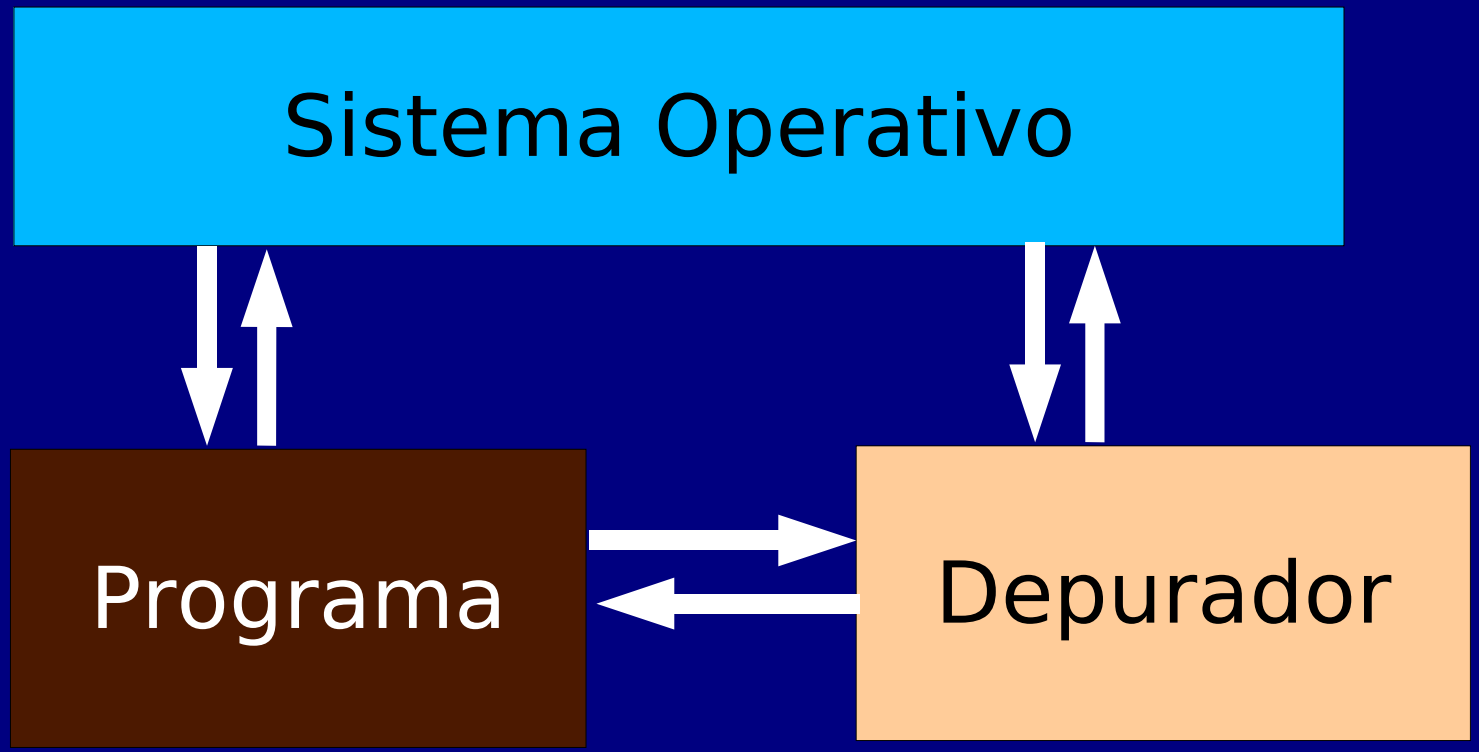
- Escribir Mensajes
  - Utilizar stderr / cerr.
  - Utilizar Macros:
    `__FILE__ / __LINE__ / __FUNCTION__`
  - Ejemplo:

```
#define DEBUG_MSG(msg, args...)                          \
        fprintf (stderr,"%s: (%s[%d]) - " msg,           \
        __FUNCTION__, __FILE__, __LINE__, ##args)
```

  - En C++, iostream.

- Un Depurador permite
  - Detener la ejecución.
  - Examinar variables.
  - Examinar las llamadas.
  - Modificar variables.
  - Llamar rutinas del programa.
  - Contar eventos.

- Cómo funciona un depurador



- Depuradores en Linux

GDB

- ¿Otros?
  - DDD – The Data Display Debugger.
  - Kdbg – Kde Debugger.
  - CGDB – Curses Frontend to GDB.
  - RHIDE / Kdevelop / etc.

- Uso de GDB
  - Pasos Previos.
  - Cargar GDB.
  - Detener el programa.
  - Examinar la pila de ejecución.
  - Examinar variables.
  - Ejecución Paso a Paso.
  - Puntos de quiebre avanzados.
  - Otros usos avanzados.

- Pasos Previos.
  - El programa del tutorial:

```
int calcula_suma( int a, int b) {
 return a+b;
}

int main(int argc, char ** argv ) {
 int suma = 0;
 while(suma < 100) {
  int n;
  printf("Ingresa un número:\n");
  scanf("%d",&n);
  suma = calcula_suma(suma, n);
  printf("La suma es hasta ahora:%d\n",suma);
 }
 printf("Se terminó.\n");
 return 0;
}
```

- Pasos Previos.
  - Compilar con opciones de depuración.

```
gcc -Wall [-O] -g -o prog prog.c
```

  - Usar Bibliotecas para Depuración.

- Cargar GDB
  - Cargar el programa

```
gdb ./prog
```

  - El entorno de GDB.

- Detener la ejecución
  - Señales Externas.
  - Los puntos de quiebre.
    break – Ingresa un punto de quiebre.
  - Señales Internas - raise()

- Examinar el entorno
  - Examinar la pila de ejecución:
    bt – backtrace.
    bt full – Información completa.
    up – Sube dentro de la pila.
    down – Baja dentro de la pila.
    info locals – Variables locales.
    list – Lista el código.
    disassemble – Lista el código ASM.

- Examinar variables
  - Examinar variables del programa:
    print – Escribe.
    display – Muestra siempre.
    info locals – Muestra locales.
    info variables – Variables.
    info address <v> – Localización.

- Ejecución Paso a Paso
  - Ejecución paso a paso del programa:
    step – Próxima línea.
    next – Salta subrutinas.
    finish – Sale de subrutina.
    advance – Continua hasta un punto.
    stepi / nexti – Paso a paso por el ASM.
    jump – Salta.

- Puntos de Quiebre
  - Manejo de los puntos de quiebre
    disable / enable – (des)habilita.
    condition – Fija una condición.
    catch throw – Atrapa excepciónes, C++.
    watch – Escritura de variables.
    awatch – Acceso a variables.

- Otros Avanzados

  thread – Cambia entre hilos.
  set var – Cambia una variable.
  return – Termina una función.

- Depuradores de Memoria
  - Memoria Dinámica.
  - ¡Punteros!
  - Datos no inicializados.

- Valgrind
  - Ejecución Controlada.
  - Recompilación dinámica.
  - Seguimiento de datos.

- Valgrind
  - Distintas fachadas:
    - Chequeo de Memoria: memcheck
    - Condiciones de Carrera: hellgrind
    - Uso de Memoria: massif
    - Rendimiento de Cache: cachegrind

- Valgrind
  - Uso en chequeo de memoria.

- Memprof
  - Chequeo de memoria perdida.
  - Uso interactivo.

- Seguidores de Llamadas
  - Interacción del programa con el S.O.
  - Uso correcto de llamadas.
  - Manejo de procesos.

- Strace
  - Intercepta las llamadas con el S.O.
  - Escribe los parámetros y resultados.
  - Uso:

```
strace [-e expr] programa [parámetros]
```

- Strace
  - Expresiones comunes:

    trace=file – Operaciones a archivos.
    trace=network – Operaciones de red.
    read=[fd] – Datos leídos de [fd].
    write=[fd] – Datos escritos a [fd].

  - Opciones comunes:

    -f – Sigue los “fork”.

- Análisis de Ejecución (excecution profile)
  - Datos estadísticos de ejecución:
    - ¿quién llamo a una función?
    - ¿cuántas veces se ejecutó el código?
    - ¿cuánto tiempo tomó?
    - ¿dónde se emplea mayor tiempo? (hot-spots)

- Análisis de Ejecución
  - ¿Cómo funciona un Analizador?
    - Compilación Instrumentada.
    - Información de Depuración.
    - Biblioteca de Recolección.
    - Herramienta de análisis de datos.

- Análisis de Ejecución
  - Compilación Instrumentada:

```
gcc -Wall [-O] -g -pg -c prog.c
   gcc -g -pg -o prog prog.o
```

- Análisis de Ejecución
  - Herramienta de Análisis:

    GPROF

- Gprof
  - Opciones de reporte:
    - Reporte normal

```
gprof prog
```

    - Reporte línea a línea

```
gprof -l prog
```

Preguntas