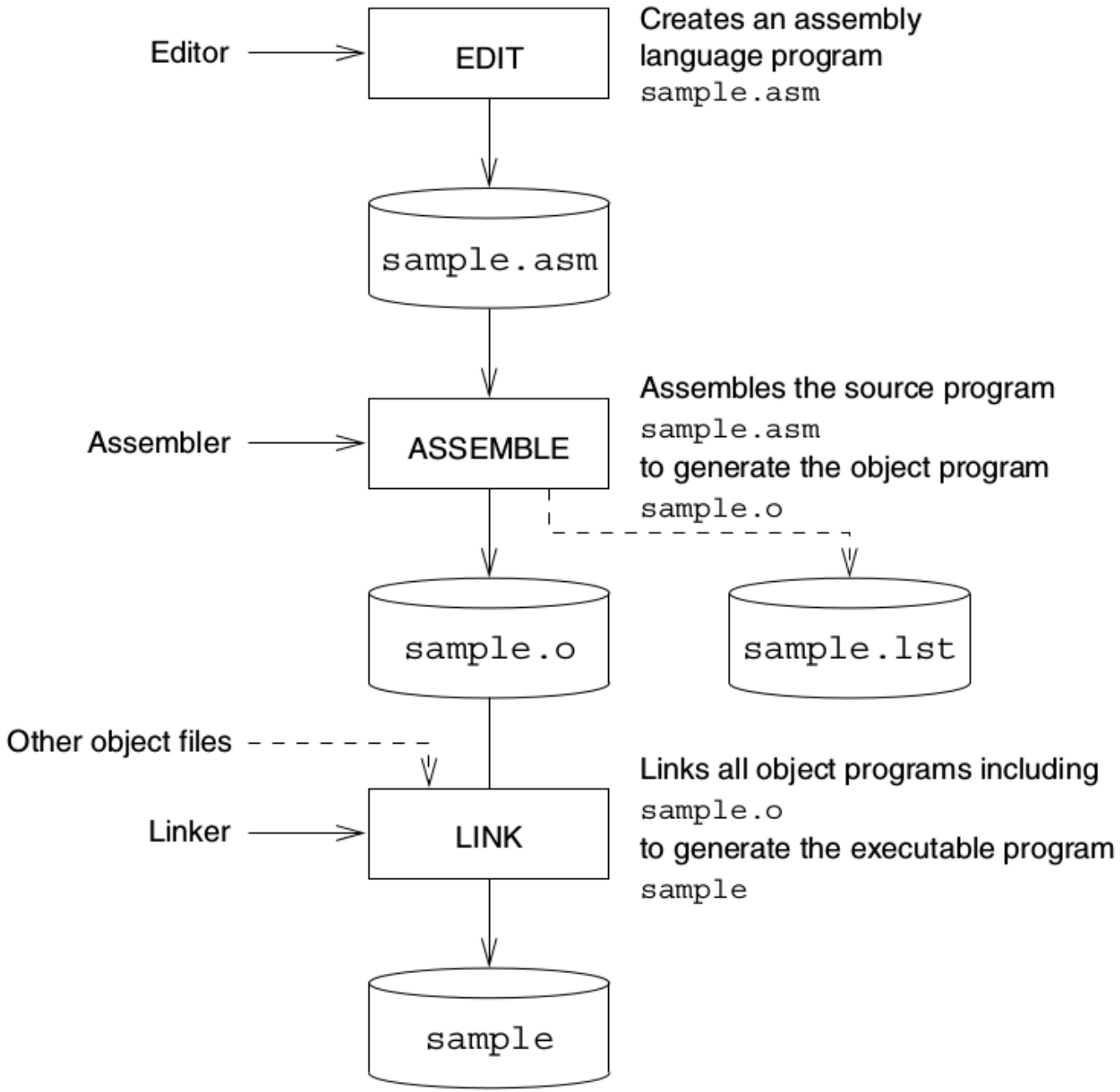
Unelte, Utilitare

Modificat: 29-Oct-18

Cuprins curs 6

- Procesul de asamblare, linkeditare
  - nasm, gcc, ld, make, libc, ldd
- Vizibilitate variabile, scoping
  - Directive de asamblare
- Objdump
- Gdb
  - .gdbinit folosite în acest curs
- nm, strings, strip
- biew, xxd, od
- bc

Asamblare, Linkeditare



De ce programe mixte?

- Avantaje si dezavantaje ale limbajului de asamblare
  - Avantaje:
    - Acces la operatii low-level
    - Performanta
    - Control asupra programului
  - Dezavantaje:
    - Productivitate scazuta
    - Greu de asigurat mentenanta
    - Lipsa de portabilitate
- Prin urmare, unele programe sunt mixte (system software)

Compilarea programelor mixte

- Putem folosi programare mixta in C si limbaj de asamblare
- Vom pune accentul pe principii
- Acestea pot fi generalizate la orice tip de programare mixta
- Pentru compilare:

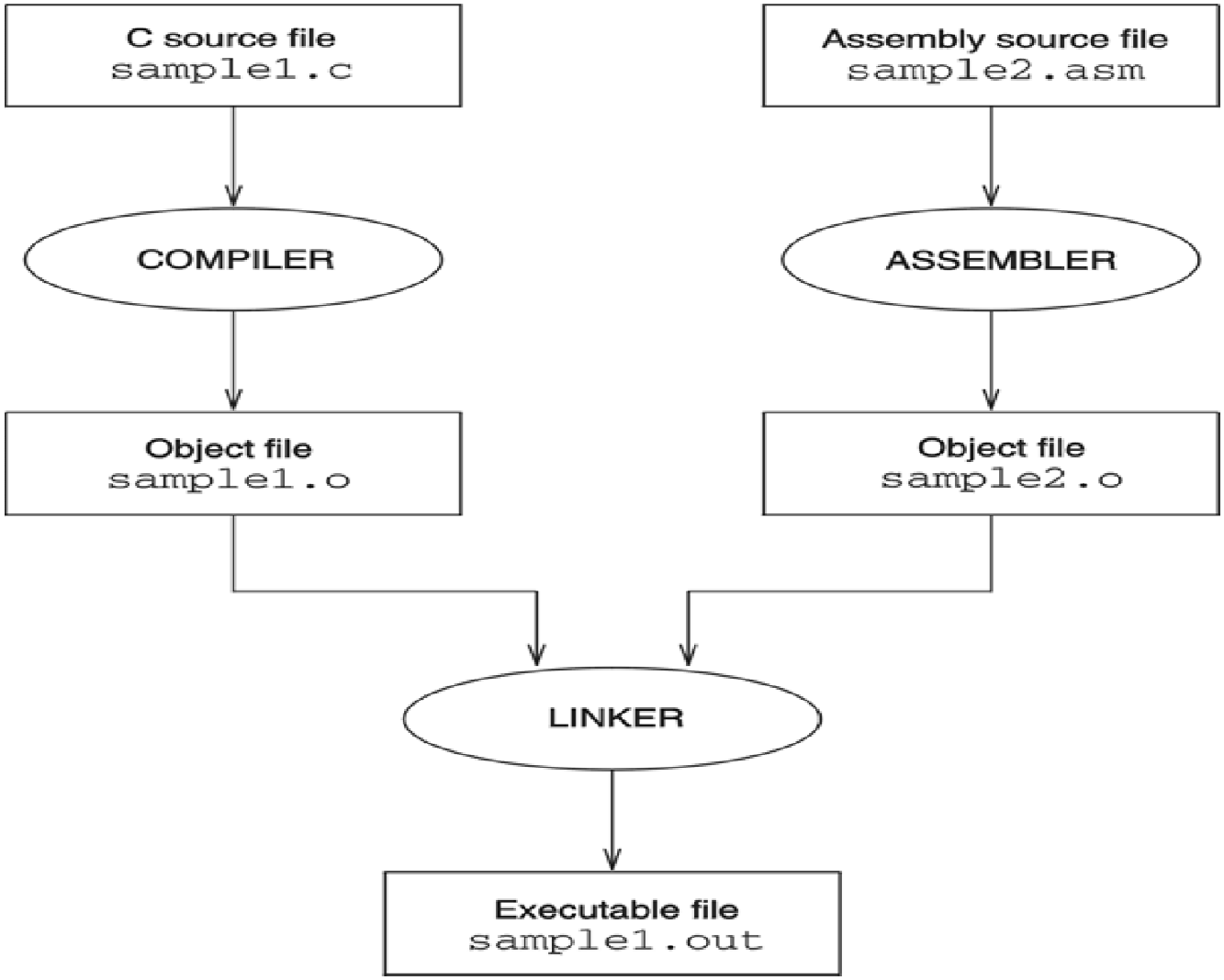
nasm –f elf sample2.asm

» creeaza **sample2.o**

gcc –o sample1.out sample1.c sample2.o

» creeaza **sample1.out** fisier executabil

Obținerea unui executabil mixt



Variabile globale

- Trei zone de memorie
  - .data: initializate
  - .bss: neinitializate (zero @load-time)
  - .rodata: initializate, read-only
- .data: int a = 10;
- .bss: int a;
- .rodata: const int a = 10;
- Variabile locale declarate **'static'** in C
- Experiment: comparați executabilele generate cu
  - **int a[10000000];**
  - **int a[10000000] = {1};**

Programe compuse din mai multe module

- Un program poate conține fișiere sursă diferite
- Avantaje
  - » Daca un modul este modificat, doar acela este recompilat (nu tot programul)
  - » Mai multi programatori pot folosi module comune
  - » Modificarea programului este mai usoara datorita fisierelor de dimensiune redusa
  - » Modificarile neintentionate pot fi evitate
- Directive de asamblare:
  - » GLOBAL si EXTERN

Scope & linkage în C

- http://norswap.com/c_scope_duration_linkage/
- Scope: domeniul de vizibiliate al unei variabile
  - block scope (locală unui bloc/funcții)
  - file scope (locală unui fișier/unități de compilare)
- Linkage: declarații multiple pentru aceeași definiție
  - no linkage (locală unui block)
  - internal linkage (globală, marcată cu static)
  - external linkage (globală, nemarcată cu static)

Directiva GLOBAL

- Directiva GLOBAL marcheaza etichetele vizibile global
  - » etichetele pot fi accesate si din alte module ale programului
- Formatul este

  global label1, label2, . . .
- Aproape orice label poate fi declarat global
  - » Nume de proceduri
  - » Nume de variabile
  - » equated labels
  - * Intr-o constructie GLOBAL, nu este necesar sa mentionam tipul labelului

Directiva GLOBAL – ex.

```
global    error_msg, total, sample
                    . . . . .
.DATA
error_msg    db    'Out of range!',0
total        dw    0
                    . . . . .
.CODE
                    . . . . .
sample:
                    . . . . .
                    ret
```

Directiva EXTERN

- Directiva EXTERN ii spune asamblorului ca anumite labeluri nu sunt definite in modulul curent
  - Asamblorul rezerva spatiu in fisierul obiect pentru a fi utilizat ulterior de linker
- Formatul este

  extern label1, label2, . . .

  unde **label1** si **label2** sunt declarate global folosind directiva GLOBAL in alte module

Directiva EXTERN(cont'd)

Exemplu – curs-07-demo

module1.asm

- Procedura main

module2.asm

- Procedura string length

Link-editare

- Rezolvarea referințelor între module, biblioteci
- Bibliotecă = colecție de fișiere .o
- Utilitar: gcc sau ld
- Link-editare
  - Statică – fișiere obiect incluse/relocate în executabilul final
  - Dinamică – referințele vor fi rezolvate la runtime
    - » Exemplu libc (printf, malloc, strcmp)
    - » **#readelf --dyn-syms hello_world** (din C)

Structura executabilului Linux

- Antet ELF #**readelf –h binar**
- Secțiuni #**readelf –S binar**
  - .text, .data, .bss, multe altele (symbols, debug)
  - **size binar**
- Instrucțiuni de încărcare #**readelf –l binar**
- Simboluri #**readelf –s binar**, #**nm –A binar/object**
- Încărcarea în memorie

https://github.com/corkami/pics/blob/master/binary/elf101/elf101.pdf

objdump

- Simboluri si segmente
  - **#objdump –t fisier.o**
- Dezasamblare
  - **#objdump –d –M intel fisier.o**
- Dezasamblare după linkeditare
  - **#objdump –d –M intel fisier_executabil**
  - Comparati adresele și destinațiile salturilor
  - Vizualizați codul binar pentru fiecare instrucțiune
  - Vizualizați valorile immediate în codul binar

curs-04-demo; #objdump –d –M intel string.o

Offset machine code str_cpy:

```
0000003b      720F          jb    4c <sc_no_string>
0000003d      89C1          mov   ECX, EAX
                              . . . .
4C= 3D+0F                   ;

                          sc_no_string:
0000004c      F9            stc
                              . . .
```

GDB

Moduri de rulare GDB

- Pornim direct programul
  - gdb ./vuln
- Ne atașăm unui progam existent
  - gdb <PID>

Comenzi importante

- b – breakpoint
- r - run
- p – print expression
- x – eXamine memory
  - sintaxa pointerilor este din C
- set – modify registers, variables
- n – next
- s – step
- q - quit
- help x – help pentru comanda x

Pornirea programului cu parametri

- run
- start
- set args 1 2 3
  - (gdb) b main
  - (gdb) b _start
- run 1 2 3
- run < file
- run $(python -c ‘print 32*”A”’)
- run < <(python -c ‘print 32*”A”’)

Afișarea de simboluri

- info registers
- p/[xdut] $ebx
  - x = hexa, d = signed, u = unsigned, t = binar
- p &buffer – diferențe C/asm
- p buffer – printează un uint32_t de la adresa buffer

Schimbare variabile/registre

- (gdb) set $eax 0x100
- (gdb) set $eip main: va continua de la acea etichetă
- (gdb) set msg “hello”: la adresa msg va scrie octeții ‘h’ ‘e’ ‘l’ ‘l’ ‘o’
- (gdb) set $x = 10: definește o variabilă în shelul de gdb

Examinare memorie

x/CFU address – count format unit

* (gdb) x/20xh &buffer: afișează în hex 20 de halfwords
* (gdb) x/50uw &buffer – 10: 50 de uint32_t de la (buffer – 40)
* (gdb) x/10cb &buffer: 10 caractere ascii
* (gdb) x/5xw $esp – 16: ultimele 4 valori de 32 biți în stivă, și spațiul ce urmează a fi folosit de următorul push

Căutarea în memorie

- Se investighează zonele procesului
  - info proc mappings
- find 0x80909090, 0x8090a000, “msg”

Breakpoints

- b system
- b *0x08043682
- info breakpoints
- d 1: șterge primul breakpoint
- c, pentru continue

Instruction Stepping

- si/stepi: nested, intră în funcții
- ni/nexti: non-nested, tratează apelurile call ca un apel oarecare

.gdbinit

- Customizare gdb, în python
- Utilizare front-end (sasm, ddd)
- Utilizare specializată peda.py (securitate)
- Exemplu vizualizare stivă (cursurile 7-9)
  - https://github.com/iocla/util/dashboard.py

NM, STRINGS, STRIP

Utilitare string-uri

- String-uri: date din .data, simboluri de compilare
- Asamblare/compilare cu –g
- #nm fisier.o: afișează simbolurile și etichetele
- #nm –A binar: afișează simbolurile
- #strip binar: aruncă simbolurile de debug
- #strings binar: tot ce e găsit asciiz în binar

XXD, BIEW

Utilitare conversii

```
# python -c "print 3*'hel\x10lo' " | xxd -g 1
0000000: 68 65 6c 10 6c 6f 68 65 6c 10 6c 6f 68 65 6c 10  hel.lohel.lohel.
0000010: 6c 6f 0a                                         lo.
```

- Binar la hex și înapoi:

```
#echo “Hello world!”  | xxd > hello.xxd
#cat hello.xxd | xxd –r
  “Hello world!”
```

- biew – utilitar de editare/dezasamblare
  - Demo
  - https://github.com/iocla/util/biew....deb

Intrebări?