Programmazione di Sistema e di Rete

Matteo Aprile
Professore: Franco Tommasi

INDICE

I. LIBRI DI TESTO CONSIGLIATI

- Advanced Programming in the Unix Environment, 3th ed, Stevens, Rago
- TCP/IP 1, Stevens (facoltativo)
- Unix Networking Programming the Socket Networking API, Stevens
- The Linux Programing Interface, Kerrisk
- manset
- Gapil Guida alla Programmazione in Linux, Simone Piccardi

II. INTRODUZIONE

SYSTEM CALL: sono uguali alle funzioni di libreria dal punto di vista sintattico, cambia il modo di compilarle. Notare che non possono essere usati i nomi delle SC per delle function call.
Per poi poter "raccontare" tra umani le sequenze di bit che vengono mandate ai processori si usa assembly.
Sono effettivamente delle chiamate a funzioni ma poi dal codice assembly puoi capire che è una system call dato che ha dei meccanismi specifici.
Alcuni esempi di chiamate e registri:

- eax : registro dove metti il numero della sc
- int 0x80: avvisa il kernel che serve chiamare una sc
- exit(): chiudere un processo
- write():

```
mov edx,4    ; lunghezza messaggio
mov ecx,msg  ; puntatore al messaggio
mov ebx,1    ; file descriptor
mov eax,4    ; numero della sc
int 0x80
```

dove nel file descriptor indichi a quale file devi mandare l'output. Questo viene usato dato che così non deve sercare il path ogni volta ma lo mantiene aperto riferendosi ad esso tramite il numero

———

```
aaprile317@hplinux3**:**~/PSR2022/apue.3e**$
     find . -type f -perm -0100
./standards/makeopt.awk
./standards/makeconf.awk
./proc/awkexample
./systype.sh
./advio/fixup.awk
```

programma Make: sulla base del file: Makefile oppure uso make -f in questo faile ci sono le regole di cosa fare in mod da automatizzare delle azioni per un n numero di file. se durante la compilaizone di massa una di queste da un errore, il programma make si interrompe, per evitare ciò si usa '-i' (ignore)
direttive di preprocessore (-DMACOS -D$_DARWIN_{CS}OURCE$) : $sonodelleindicazionifatteprimadiiniziarelacompilaizione, include)includepuoaverelibreiredisistema < lib > oppuredilibreriefattedanoienonindirectorystandard$
altre: define:
sostituzione di una stringa con un'altra (detta "macro") (es: define BUFLEN), utile per ifdef. possono essere passate a linnea di comando le macro per poterle definire function like macro (define ABSOLUTE$_VALUE(x)(((x) < 0)? - (x) : (x)))doveemetteràl'arbomentoalpostodellaxifdef, ifndef, endif$
es una macreo è stata definita allora fai ciò che segue: ifdef VAR print("hello"); endif per evitare che più file includano lo stesso si usano degli ifndef in tutto il codice, in modo da evitare doppie definizioni
nella compilaizone creiamo dei file oggetto per ogni file .c (gcc -c bill.c) creando dei file .o (oggetto). un file compilato è un file dove ci son otanti bit che sifgnificano qualcosa per il processore. si va quindi a creare il prototipo della funzione. tramite il linker si andranno ad unire tutti i file per crearne uno unico con tutto al suo interno(tutte le chiamate) scioglimento dei riferimenti incrociati. dove quandi si vanno a recuperare le definizioni
file dei protipi .h file delle definizioni .c
abbiamo già le istruzioni del processore per le funzioni di sistema quindi abbiamo il file sorgente si e no dato che abbiamo direttamente l'eseguibile. jquesto codice si trova in un file di libreia cioè un insieme di file oggetto linkati in un unico file. al suoi interno c'e i lcodice ogetto di tutte le funzioni
libreire dinamiche, stessa cosa ma stanno epr fatti loro (dormienti) e vengono linkate a runtime all'occorrenza
per fare il linking: gcc -o program program.o bill.o

III. 28.09.2022

parliamo di librerie: abbiamo example file e la pagina di istruzioni, dove abbiamo un file di intestazione. obbiettivo: distribuire una libreria. per costruirla dal file example abbiamo: static lib: stesso procedimento per linux e macos dynamic: cambiano
static: collezione di file oggetto che hanno il codice compilato delle funzioni questi file vengono linkati al momento della compilazione. quindi si fondono con li codice diventando un unico eseguibile che è un programma che chiunque con lo stesso OS può eseguire. il difetto è che a volte si aggiornano le librerie al momento del ritrovamento di un bug. se il bug viene corretto se hai il file già compilato lo avrai comunque buggato e quindi servirà ricevere la versione corretta ed aggiornata dynamic: ricordano il concetto di plug in, quindi viene invocato a runtime il caricamento nella memoria. in genere questo si fa con gli aggiornamenti dei OS sempre per correggere dei bug. l'eseguibile quindi non viene toccato la correzione avviene solo nella libreria. il difetto è che finche sono librerie di sistema il requisito maggiore è che chi si passa il codice debba avere lo stesso OS dell'altro utente. notare che non cambia il prototipo dato che sennò bisognerà ricompilare il programma e questo non va bene
nel questo generale, le librerie statiche sono molto pericolose infatti alcuni OS le aboliscono per le questioni di sistema, dato che le lib.c che è la libreria con le funzioni più usate in c, esiste come libreria statica su linux con estensione .a invece per macos è stata abolita dato che non vogliono che si corra il rischio di sicurezza.per dire se si vuole compilare con la versione dinamica o statica si usa in gcc l'opzione -static.
costruiamo una libreria statica, andando a distribuire la libreria ma non il suo codice: MACOS: 1. costruiamo il file oggetto: gcc -c libprova.c 2. costruiamo la libreira: ar (archive) rcs (c sta per create se la libreria .a non esiste) libprova.a libprova.o 3. costruire il codice che usa la libreria: gcc -Wall (verbose dei warning) -g (permette i ldebugging del codice e vedere l'istruzione di c alla quale sei) -c (crea il file) useprova.c 4. link che riso;ve le chiamate incrociamte tra file sorgente e libreire: gcc -g -o useprova useprova.o -L. (dove prendere la libreira che serve) -lprova (per usare la libreria)
sezioni del manuale: 1: funzioni che possono essere usate a linea di comando 2: system call 3: libreria di sistema
per capire che librerie usa il codice si usa: otool -L [nomecodice] su linux invece: ldd [nomevodice]
LINUX: 1. costruiamo il file oggetto: gcc -fPIC -Wall -g -c libprova.c 2. costruiamo la libreira: gcc -g -shared -Wl,-soname,libprova.so.0 -o libprova.so.0.0 libprova.o -lc (indica che usa libc) 3. costruire del link simbolico usato per aggiornare le librerie senza aggiornare gli eseguibili. quindi si puo usare sempre lo setsso nome del programa ma saprà lui grazie ai link simboliic lo porterrano al file con la versione giusta: ln -sf libprova.so.0.0 libprova.so.0 4. link che riso;ve le chiamate: ln -sf libprova.so.0 libprova.so
per defaoult avremko una compilazione dinamica, in caso constrario possiamo aggiungere -static
su linux allora creeremp una variabile di ambiente imposta all'eseguibile (dato che sennò non sa dove trovarla infatti fa -¿ not found), questa variabile è $\mathrm{LD}_L IBRARY_P ATH =$ ‘$pwd‘ldduseprovaquindissesiteunalibreirasatticaluiprenderàladinar$
MACOS: la libreria dinamica è: gcc -dynamiclib libprova.c -o libprova.dylib Builds the shared library
eseguendo il programam trova la libreira dato che va a controllare nella directory corrente e quindi non serve creare la variabile di ambiente come si linux
i file di intestazione del mac come stdio.h per cercarla uso: find /Applications/Xcode.app/ -name stdio.h 2¿/dev/null
MAKE: andiamo a guardar cosa contine make file per carpire cosa fa:
DIRS = lib intro sockets advio daemons datafiles db environ fileio filedir ipc1 ipc2 proc pty relation signals standards stdio termios threadctl threads printer exercises
all: for i in $(DIRS); do$ $(cdiecho"makingi"$(MAKE)) —— exit 1; done
clean: for i in $(DIRS); do$ $(cdiecho"cleaningi"$(MAKE) clean) —— exit 1; done
DIRS è tutta quella cosa
i make file hanno delle loro regole per fare delle cose. per esempio all è detto target, cioè la cosa che vuoi fare qundi sarebbe make all. che è la cosa fi defalult dato che è il primo allora scrivendo solo make lo avvieamo. a volte possono essere dei prerequisiti che posson essere a lor volta degli altri target. la figa delle regole è quella che viene dopo i : che vengono però dopo i prerequisiti ed indica che poer fare il target bisogna usare queste regole. i vincoli sintattici sono il tabulatore e poi si mettono i programmi da shell.

RIFERIMENTI BIBLIOGRAFICI

[1] https://en.wikibooks.org/wiki/LaTeX/Hyperlinks