B

Ingegneria degli Algoritmi (A.A. 2010-2011)

Corsi di Laurea in Ingegneria Informatica e Automatica, Ingegneria dei Sistemi Informatici, e Laurea Magistrale in Ingegneria Informatica

Sapienza Università di Roma

Primo appello (15/6/2011) – Durata 2 ore – 5/6 cfu – Compito B

Cognome: _____ Nome: _____ Matricola: _____	Autorizzo la pubblicazione del voto di questo esame sul sito web http://www.dis.uniroma1.it/~demetres/didattica/ae, secondo quanto prevede il decreto legislativo 196/2003 (codice in materia di protezione dei dati personali) che dichiaro di conoscere. In fede, _____

Rispondere alle seguenti domande, **motivando le risposte** (risposte non motivate saranno considerate nulle).

Domanda 1 [8 punti]

Si consideri il seguente report di profiling dell'esecuzione di un programma che invoca le tre funzioni `main`, `f` e `g`:

- `main`: 20%
- `f`: 40%
- `g`: 40%

Dopo aver applicato un certo numero di ottimizzazioni, riusciamo a ottenere uno speedup 4x sul tempo di esecuzione della funzione `f`. Rimpiazzando `f` con la sua versione ottimizzata:

1. che speedup otterremmo per l'esecuzione del programma?
2. che report di profiling ci aspetteremmo?

Domanda 2 [10 punti]

Si consideri un allocatore di memoria di tipo *sequential fits* con politica *best-fit* che:

- non faccia splitting/coalescing
- lavori su una piattaforma a 32 bit
- allinei l'indirizzo iniziale del payload di ogni blocco a un multiplo di 4 byte
- abbia per ogni blocco una header di 4 byte contenente la size del blocco stesso

Illustrare la struttura dell'heap passo-passo durante la seguente sequenza di operazioni, a partire da un heap vuoto:

1. *inizializzazione allocatore*
2. `b10=malloc(10)`
3. `b15=malloc(15)`
4. `b06=malloc(6)`
5. `b24=malloc(24)`
6. `b05=malloc(5)`
7. `free(b24)`
8. `free(b15)`
9. `free(b05)`
10. `b13=malloc(13)`
11. `b26=malloc(26)`

Includere nei disegni i blocchi comprensivi di header, puntatori, padding, ecc. evidenziando blocchi liberi, blocchi in uso e liste di blocchi liberi.

Si consideri lo stato dell'allocatore alla fine della sequenza e si risponda alle seguenti domande:

- Si ha frammentazione esterna, interna, entrambe, o nessuna?
- Di quanti byte in avanti risulta spostato il `brk` rispetto alla base dell'heap?
- Qual è il fattore di utilizzo dell'allocatore (rapporto tra payload totale e memoria richiesta al sistema operativo mediante `sbrk`)?

Domanda 3 [8 punti]

Si consideri il seguente frammento di codice assembly[1] in sintassi AT&T generato dal compilatore `gcc` su una piattaforma x86-64 conforme alla System V AMD64 ABI:

```
f:
        pushq %rbp
        movq %rsp, %rbp
        movl (%rdi), %edx
        movl (%rsi), %ecx
        movl %ecx, (%rdi)
        movl %edx, (%rsi)
        leave
        ret
```

Rispondere alle seguenti domande.

1. Scrivere una funzione `f` in linguaggio C la cui compilazione con `gcc -S` fornisca il codice assembly sopra riportato.
2. Quale rilevante ottimizzazione è stata applicata dal compilatore nel produrre il codice assembly della funzione `f`?
3. Come ulteriore ottimizzazione, quali istruzioni x86-64 possono essere rimosse dal codice assembly della funzione `f` senza alterarne il funzionamento?

---

[1] Consultare la tabella in calce al compito per una descrizione delle istruzioni x86-64 utilizzate.

Domanda 4 [6 punti]

Sia `a` una variabile di tipo `int` e sia `b` una variabile di tipo `int*`. Per ciascuna delle seguenti espressioni C dire: 1) se sono valide; 2) che tipo hanno; 3) se denotano lvalue o rvalue.

1. `a+((int (*)(int*))b)(b+1)`
2. `a+&(b+1)`
3. `a+*(b+1)`
4. `*(a+*&b)`
5. `*(int**)&a`
6. `a+*(((struct { int c, d; }*)b)->c)`

prefix	description	example	C analog
`add`	add source to destination	`addl $5,%ecx`	`ecx += 5`
`call`	procedure call	`call _foo`	`foo()`
`cltq`	sign-extend `eax` to `rax`	–	–
`dec`	decrement destination	`decq %rcx`	`rcx--`
`imul`	multiply destination with source	`imull %esi,%eax`	`eax *= esi`
`inc`	increment destination	`incl %ecx`	`ecx++`
`ja`	jump if above *(unsigned comparison)*	`cmpl %eax,%ebx` `ja L2`	`if ((unsigned)eax > (unsigned)ebx) goto L2`
`jae`	jump if above or equal *(unsigned comparison)*	`cmpl %eax,%ebx` `jae L2`	`if ((unsigned)eax >= (unsigned)ebx) goto L2`
`jb`	jump if below *(unsigned comparison)*	`cmpl %eax,%ebx` `jb L2`	`if ((unsigned)eax < (unsigned)ebx) goto L2`
`jbe`	jump if below or equal *(unsigned comparison)*	`cmpl %eax,%ebx` `jbe L2`	`if ((unsigned)eax <= (unsigned)ebx) goto L2`
`je`	jump if equal	`cmpq %rax,%rbx` `je L2`	`if (rax == rbx)` `goto L2`
`jg`	jump if greater *(signed comparison)*	`cmpq %rax,%rbx` `jg L2`	`if (rax > rbx)` `goto L2`
`jge`	jump if greater or equal *(signed comparison)*	`cmpq %rax,%rbx` `jge L2`	`if (rax >= rbx)` `goto L2`
`jl`	jump if less *(signed comparison)*	`cmpq %rax,%rbx` `jl L2`	`if (rax < rbx)` `goto L2`
`jle`	jump if less or equal *(signed comparison)*	`cmpq %rax,%rbx` `jle L2`	`if (rax <= rbx)` `goto L2`
`jmp`	unconditional jump	`jmp L2`	`goto L2`
`jne`	jump if not equal	`cmpq %rax,%rbx` `jne L2`	`if (rax != rbx)` `goto L2`
`jnz`	identical to `jne`	–	–
`jz`	identical to `je`	–	–
`lea`	copy address to destination (load effective address)	`leaq -12(%rax),%rcx`	`rcx=rax-12`
`leave`	pop the current stack frame, and restore the caller's frame	–	–
`mov`	copy data from source to destination	`movq $7,(%rax)`	`*(long*)rax=7`
`movabs`	copy 64-bit immediate to destination register	`movabsq $-7,%rax`	`rax=-7`
`movsl`	copy sign-extended word to destination register	`movslq %bx,%rax`	`rax=bx`
`movzl`	copy zero-extended word to destination register	`movzlw %bx,%eax`	`eax=(unsigned)bx`
`movsb`	copy sign-extended byte to destination register	`movsbq %bl,%rax`	`rax=bl`
`movzb`	copy zero-extended byte to destination register	`movzbq %bl,%rax`	`rax=(unsigned)bl`
`pop`	pop value from stack and write it to destination	`popq %rbx`	–
`push`	push value on stack	`pushq %rbx`	–
`ret`	return from procedure call	`ret`	`return`
`sub`	subtract source from destination	`subl $5,%ecx`	`ecx -= 5`

Tabella 1: Istruzioni x86-64 più comunemente utilizzate.