

*Laboratorio di Sistemi Operativi
Anno Accademico 2006-2007*

AMIKaya: criteri di valutazione del progetto

Mauro Morsiani

Copyright © 2007 Renzo Davoli, Alberto Montresor, Mauro Morsiani
Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license can be found at: <http://www.gnu.org/licenses/fdl.html>

Esame – progetto per LSO

Progetto AMIKaya

Prevede lo sviluppo di alcune delle parti più significative di un moderno sistema operativo

Dovrà essere scritto in linguaggio C

Verrà testato utilizzando un simulatore (uMPS)

© 2007 Renzo Davoli, Alberto Montresor, Mauro Morsiani

2

Esame – progetto per LSO

Struttura del progetto

AMIKaya è diviso in due fasi:

phase1: introduzione all'uso del simulatore e sviluppo di moduli in C (queue manager)

phase2: sviluppo del nucleo del SO (scheduler, exception handler, primitive di sincronizzazione)

© 2007 Renzo Davoli, Alberto Montresor, Mauro Morsiani

3

Esame – progetto per LSO

Date di consegna delle fasi del progetto

phase1 "anticipata": 20/04/2007

(phase1 +) phase2 "per dare l'esame a giugno": 08/06/2007

(phase1 +) phase2 "per dare l'esame a luglio, settembre o febbraio": 06/07/2007

© 2007 Renzo Davoli, Alberto Montresor, Mauro Morsiani

4

Esame – progetto per LSO

Punteggio massimo assegnabile alle varie fasi del progetto:

punteggio massimo totale: 110 punti

phase1 “anticipata”: max 40 punti

phase1 consegnata insieme a phase2: max 30 punti

phase2: max 70 punti

Consegnare in anticipo aumenta il punteggio potenziale e diminuisce il lavoro da fare a maggio/giugno

© 2007 Renzo Davoli, Alberto Montresor, Mauro Morsiani

5

AMIKaya – realizzazione del progetto

AMIKaya deve essere strutturato in moduli:

In C non esistono gli oggetti

Alcuni concetti della programmazione ad oggetti (p.es. il *data hiding*) si possono implementare anche in C utilizzando i *moduli*

modularizzare il codice permette anche di ridurre la complessità dei problemi da risolvere

© 2007 Renzo Davoli, Alberto Montresor, Mauro Morsiani

6

AMIKaya – realizzazione del progetto

Struttura di un modulo C (1/2):

Ogni modulo è composto da un *file di implementazione* (*nome_del_modulo.c*) e da un *file di interfaccia* (*nome_del_modulo.h* oppure *nome_del_modulo.e*)

I due file devono avere lo stesso nome, cambia l'estensione

Il file di implementazione deve contenere:

l'inclusione degli header (file di libreria, etc.: tipicamente con estensione *.h*)

inclusione dei file di interfaccia degli altri moduli utilizzati

definizione di macro, tipi variabili da esportare (pubblici)

definizione di macro, tipi variabili locali al modulo

definizione di funzioni da esportare (pubbliche)

definizione di funzioni locali al modulo

© 2007 Renzo Davoli, Alberto Montresor, Mauro Morsiani

7

AMIKaya – realizzazione del progetto

Struttura di un modulo C (2/2):

Il file di interfaccia deve contenere:

definizione di macro, tipi da esportare (pubblici)

dichiarazione `extern` delle variabili da esportare

dichiarazione dell'interfaccia delle funzioni da esportare

Potranno esistere dei file di inclusione (*file_da_includere.h*) non correlati a singoli moduli

Questi file devono contenere costanti, tipi e macro globali rispetto all'intero progetto, e non relativi a un modulo specifico

In AMIKaya l'estensione per il file di interfaccia sarà `.e`

© 2007 Renzo Davoli, Alberto Montresor, Mauro Morsiani

8

AMIKaya – realizzazione del progetto

Valutazione del progetto (1/2):

La valutazione del lavoro terrà conto dei seguenti aspetti:

Struttura (architettura, ingegnerizzazione) del programma:

- sviluppo top-down
- divisione in moduli
- linearità e corretta definizione delle funzioni
- mancanza di codice ripetuto
- scope delle variabili globali
- mancanza di side-effect per le funzioni
- architettura all'interno del file system
- makefile

© 2007 Renzo Davoli, Alberto Montresor, Mauro Morsiani

9

AMIKaya – realizzazione del progetto

Valutazione del progetto (2/2):

Leggibilità del codice:

- documentazione (interna ed esterna)
- chiarezza dichiarativa del codice (uso di costanti, scelta dei nomi per le costanti, le variabili, le funzioni, le macro)
- indentazione del codice

Coerenza con le specifiche di funzionamento

Gestione di casi particolari e/o loro discussione nella documentazione

Funzionamento corretto

Efficienza

© 2007 Renzo Davoli, Alberto Montresor, Mauro Morsiani

10

AMIKaya – realizzazione del progetto

Documentazione interna: i commenti nel codice

In italiano o in inglese (italiano corretto, o inglese leggibile)

Inserire la quantità *minima* di commenti che renda chiaro il funzionamento del programma:

tanti commenti = costo (+ commenti = + tempo per scriverli e per leggerli)

pochi commenti = costo (- commenti = + tempo per capire il programma)

Valutazione commenti: non in base a quantità, ma in base a qualità

Commenti interni:

- uno all'inizio del file, per descrivere quali caratteristiche accomunano le funzioni del modulo
- uno prima di ogni funzione, per descrivere che cosa la funzione fa
- uno prima di ogni punto "oscuro"

Commenti esterni: file .txt o .html

© 2007 Renzo Davoli, Alberto Montresor, Mauro Morsiani

11

AMIKaya – realizzazione del progetto

CODICE SOTTOCOMMENTATO

```
/* elabora fft se $1 ==
3 */
main(int argc, char *
argv[])
{
if atoi(argv[1]==3)
fft(v);...
```

```
/* Programma FFT:
Parametri: * nro di
dimensioni... */
main(int argc, char *
argv[])
{
if atoi(argv[1]==3)
/*caso a 3 dimensioni*/
fft(v);...
```

CODICE SOVRACOMMENTATO

```
/* se il primo parametro che
viene passato al programma in
argv[1] è 3 allora viene
calcolata la trasformata rapida
discreta di Fourier sul vettore
v */
main(int argc, char * argv[])
/*argc == n.ro argomenti*/
{
if atoi(argv[1]==3) /*caso a 3
dimensioni */
fft(v);
```

© 2007 Renzo Davoli, Alberto Montresor, Mauro Morsiani

12

AMIKaya – realizzazione del progetto

Errori: alcuni esempi

Gravissimi:

`goto`, `break` all'interno di cicli

Gravi:

mancanza di `makefile`

`main()` "che fanno tutto"

`makefile` usato come shell script

presenza di side-effect

Altri:

mancanza di coerenza fra le parti realizzate dai vari componenti del gruppo

parti sotto-commentate

ripetuti errori di ortografia nei commenti

© 2007 Renzo Davoli, Alberto Montresor, Mauro Morsiani

13

AMIKaya – realizzazione del progetto

Progetto AMIKaya: testing

Per ogni fase verrà distribuito un programma di alpha-testing (`p1test.c`, `p2test.c`) che permetterà di verificare il corretto funzionamento dei moduli sviluppati

Se il programma di test non termina correttamente, i moduli contengono (molto probabilmente) qualche errore

Il fatto che il test venga eseguito in maniera positiva però *non garantisce* che i moduli siano veramente corretti

© 2007 Renzo Davoli, Alberto Montresor, Mauro Morsiani

14

AMIKaya – realizzazione del progetto

Progetto AMIKaya: consegna

Occorre consegnare un file di tipo `tar.gz` (tar compresso) contenente:

sottoalbero del file system che contenga la "distribuzione" del lavoro fatto, con unico punto di ingresso

nella root dir della distribuzione deve essere presente un file `README` con:

istruzioni su come compilare ed eseguire il lavoro

scelte progettuali globali

documentazione in formato `txt` o `html`

nella root dir della distribuzione deve essere presente un file `AUTHORS` contenente i nomi degli autori (non riportateli in ogni file)

il o i `makefile` per la compilazione del tutto

devono descrivere cosa stanno facendo

devono contenere un target `clean`

non devono eseguire il codice, se mediante un target separato es. `test`

La consegna dovrà essere effettuata in una apposita directory che verrà creata a questo scopo e comunicata a lezione e sul newsgroup

© 2007 Renzo Davoli, Alberto Montresor, Mauro Morsiani

15

AMIKaya – realizzazione del progetto

Progetto AMIKaya: phase1 e phase2

E' perfettamente lecito modificare `phase1`, dopo averla consegnata anticipatamente, per sviluppare `phase2`

Attenzione però: `phase1` viene valutata solo la prima volta

Quando consegnate `phase2`, non dimenticate di ri-accludere `phase1`, se l'avete consegnata anticipatamente

Il `makefile` di `phase2` deve permettere di compilare `phase1` e `phase2` separatamente

La documentazione per `phase1` e `phase2` deve essere separata

Impostate da subito lo sviluppo del progetto (albero di distribuzione, `makefile`, directory per file comuni, documentazione, etc.) tenendo conto di queste esigenze

© 2007 Renzo Davoli, Alberto Montresor, Mauro Morsiani

16

AMIKaya – realizzazione del progetto

Progetto AMIKaya: cosa non consegnare

Non consegnare:

dischetti / CD / DVD

materiale cartaceo

via mail, news, o sulla vostra home page

il vostro lavoro rischierebbe di non essere valutato

Non consegnare:

file eseguibili, file di prova, materiale non correlato all'esercizio

Consegnate, invece, anche se il vostro lavoro non funziona perfettamente, o non funziona affatto

La progettazione viene comunque valutata, la mancata consegna impedisce l'accesso all'esame