# lso13az09 - Documentazione fase 1 progetto SOS 2012/2013

Authors:
Orgest Shehaj <shehaj@cs.unibo.it>
0000577768
Fabian Priftaj <priftaj@cs.unibo.it>
0000355667
Eduart Uzeir <uzeir@cs.unibo.it>
090020667

# Indice

# 1 Phase1

La prima fase del progetto SOS (Strange Operating System) consiste nella realizzazione del livello due del sistema operativo ideato da Dijkstra, nel modello a 7 livelli, nel livello due si gestiscono le code dei processi in esecuzione nel sistema operativo.

La prima fase si compone in due moduli, Process Control Block (PCB), ed Active Semaphore List (ASL)

## 1.1 pcb.c

In questo modulo vengono gestite tre prati diverse del sistema operativo. Una parte relativa all'allocazione e alla deallocazione, una parte relativa alle code dei processi e una terza parte che gestisce gli alberi.

Ogni parte contiene delle funzioni che compiono delle semplici operazioni sulle liste.

## 1.2 asl.c

Un semaforo è un tipo di dato astratto , gestito da un sistema operativo multitasking per sincronizzare l'accesso a risorse condivise tra task.

# 2 Compilazione esecuzione di phase 1

Per compilare i vari moduli ed il test si utilizza il comando make.

Esempio:

$ cd /SOS (enter)

$ make (enter)

Dopo aver inserito questi comandi il progetto verrà compilato e i file oggetto si creano automaticamente.

Dopo di chè si può procedere con il lancio del simulatore uMPS2, come segue...

$ umps2 (enter)

uMPS2 procederà con le sue fasi di inizializzazione chiedendo un file di configurazione che si crea automaticamente specificando la directory dove si vuole creare questo file e poi, usando la GUI del simulatore si esegue tutto.

Dopo premendo Alt+0 si aprira una finsetra del simulatore che elencerà il procedimento del test. In fine dopo che tutti i testo sono stati passati si puo chiudere il simulatore e si puo performare il comando per cancellare i file di compilazione.

$ make clean (enter)

Questo comando cancellerà tutti i file oggetto e nelle directory rimarranò solo i file sorgenti.

La prima fase di questo progetto prevede la definizione di alcune strutture dati e metodi necessari per sviluppare la successiva fase di creazione delle principali componenti di un Sistema Operativo ad architettura a MicroKernel.

# 3 Gestione dei pcb e delle code

## 3.1 Gestione dei pcb

I pcb sono gestiti usando i concetti di coda.

Tutti i pcb sono mantenuti in una struttura pcbFree_Table che fisicamente si presenta come un array, ma logicamente permette una gestione avanzata con code .

Le code sono popolate da strutture pcb_t, composte da 7 campi: che servono per la gestione dei processi.

void initPcbs(void):

Inizializza la pcbFree in modo da contenere tutti gli elementi della pcbFree_table.

void freePcb(pcb_t *p):

Inserisce il PCB puntato da p nella lista dei PCB liberi

pcb_t *allocPcb(void):

Restituisce NULL se la pcbFree e' vuota. Altrimenti rimuove un elemento dalla pcbFree, inizializza tutti i campi (NULL/0) e restituisce l'elemento rimosso.

void insertProcQ( pcb_t **head, pcb_t* p):

inserisce l'elemento puntato da p nella coda dei processi puntata da head. L'inserimento deve avvenire tenendo conto della priorita' di ciascun pcb

pcb_t* headProcQ(pcb_t* head):

Restituisce l'elemento di testa della coda dei processi da head .

pcb_t* removeProcQ( pcb_t** head):

rimuove il primo elemento dalla coda dei processi puntata da head.

pcb_t* outProcQ( pcb_t** head, pcb_t *p):

Rimuove il PCB puntato da p dalla coda dei processi puntata da head

void forallProcQ(struct pcb_t *head, void fun(struct pcb_t *pcb, void *), void *arg):

richiama la funzione fun per ogni elemento della lista puntata da head

void insertChild(pcb_t *parent, pcb_t *p):

Inserisce il PCB puntato da p come figlio del pcb puntato da parent.

pcb_t* removeChild(pcb_t *p):

Rimuove il primo figlio del PCB puntato da p. Se p non ha figli, restituisce NULL.

pcb_t * outChild(pcb_t* p):

Rimuove il PCB puntato da p dalla lista dei figli del padre. Se il PCB puntato da p non ha un padre, res*tuisce NULL. Altrimenti restituisce l'elemento rimosso (cioe' p). A differenza della removeChild, p puo' trovarsi in una posizione arbitraria .

## 3.2 Gestione delle code

In SOS si gestisce l'accesso alle risorse da parte di più processi contemporaneamente attraverso l'utilizzo dei semafori. I semafori sono realizzati attraverso

una struttura dati che fa da descrittore (semd), la quale contiene i puntatori necessari a collegare il semaforo.

In particolare utilizziamo due code. Una coda e utilizzata per la gestione dei pcb occupati e una per i pcb liberi.

semd_t* getSemd(int *key):

Cerca il semaforo con chiave key

void initASL():

Inizializza la lista dei semdFree in modo da contenere tutti gli elementi della semdTable.

int insertBlocked(int *key, pcb_t* p):

Viene inserito il PCB puntato da p nella coda dei processi bloccati associata al SEMD con chiave key. Se il semaforo corrispondente non e' presente nella ASL, alloca un nuovo SEMD dalla lista di quelli liberi (semdFree) e lo inserisce nella ASL, settando I campi in maniera opportuna. Se non e' possibile allocare un nuovo SEMD perche' la lista di quelli liberi e' vuota, restituisce TRUE. In tutti gli altri casi, restituisce FALSE.

pcb_t* removeBlocked(int *key):

Ritorna il primo PCB dalla coda dei processi bloccanti (s_ProcQ) associata al SEMD della ASL con chiave key.

pcb_t* outBlocked(pcb_t *p):

rimuove il pcb puntato da p dalla coda dei processi

pcb_t* headBlocked(int *key):

restituisce il puntatore al pcb del primo processo bloccato sul semaforo, senza deaccordarlo.

void outChildBlocked(pcb_t *p):

Rimuove il PCB puntato da p dalla coda del semaforo su cui e' bloccato , termina anche tutti i processi discendenti

void forallBlocked(int *key, void fun(struct pcb_t *pcb, void *), void *arg):

richiama la funzione fun per ogni processo bloccato sul semaforo identificato da key.

# 4 Correzioni

Non sono state fatte correzioni ai files delle specifiche .

# 5 Directory

I file e le direcotory sono organizzate nel modo seguente :

./Doc/ Contiene la documentazione, il file Authors, il Readme, la licenza.

./src/ Contiene a sua volta tre cartelle, include, modules e phase1

./src/include/ Contiene i file di header

./src/modules Contiene i file eiffel

./src/phase1 Contiene pcb.c, asl.c e p1test.c

## 6 Source code