Scuola di Scienze Matematiche, Fisiche e Naturali
Corso di Laurea in Informatica

Tesi di Laurea

TITOLO ITALIANO

TITOLO INGLESE

NOME CANDIDATO

Relatore: *Relatore*
Correlatore: *Correlatore*

Anno Accademico 2014-2015

INDICE

ELENCO DELLE FIGURE

"Inserire citazione"
— Inserire autore citazione

1

INTRODUZIONE A SEL4A

Un sistema operativo (SO) è un insieme di software che gestisce le risorse hardware e software di un sistema di elaborazione fornendo servizi agli applicativi utente.
In un computer quindi il sistema operativo fornisce l'unica interfaccia diretta con l'hardware e in quanto tale ha un accesso esclusivo con il massimo dei privilegi chiamato *kernel mode*. Questo comporta che una vulnerabilità all'interno del sistema operativo può portare a gravi conseguenze per l'integrità e la sicurezza del sistema e dei dati che gestisce, in quanto qualche malintenzionato potrebbe approfittare di questo bug per trarne profitto. Uno degli obiettivi principali di un SO è quindi quello di garantire la sicurezza, un altro importante ... è l'efficienza: un buon sistema operativo deve saper sfruttare al meglio tutte le risorse che ha a disposizione, dalla gestione della memoria per sfruttare al meglio lo spazio alla schedulazione dei processi per ottimizzare i tempi di esecuzione. Come ultimo obiettivo, ma non per questo meno rilevante, deve rendere il più semplice possibile l'utilizzo del dispositivo su cui è installato.
... argomentare ovviamente

1.1 MICROKERNEL E KERNEL MONOLITICI

Esistono vari modelli strutturali per i sistemi operativi: monolitici, modulari, a livelli, microkernel ed ibridi, ad oggi i più diffusi sono gli ibridi, che combinano i vari modelli tra di loro, ma che in gran parte si basano su sistemi monolitici i quali consistono di un unico file binario statico al cui interno sono definite tutte le funzionalità del kernel e che viene eseguito in un unico spazio di indirizzi, questo comporta dei vantaggi e degli svantaggi: un grosso punto a favore è l'efficienza, lavorando nello stesso spazio di indirizzi e gestendo tutto attraverso chiamate di sistema il SO risulterà molto reattivo e performante, (Svantaggio che il SO è gros-

so e un bug rompe tutto e non è possibiole aggiungere moduli) d'altra parte ha anche degli svantaggi: inserire un nuovo servizio richiede la ricompilazione del kernel; un altro grosso svantaggio è la dimensione, dovendo gestire tutte le principali funzionalità del sistema operativo il kernel sarà composto da milioni di righe di codice (MLOC - linux ha circa 20MLOC) che ha come conseguenza un ulteriore problema: maggiore è il numero di righe di codice maggiore sarà il numero di possibili bug, essendo tutto il codice eseguito nello stesso spazio di indirizzi un bug rischia di far bloccare l'intero sistema anche se il problema fosse molto piccolo e isolato a una minima funzione del kernel.
(entrare un pò più nello specifico?)
All'estremo opposto troviamo i *microkernel* che sono composti da un kernel ridotto al minimo indispensabile mentre tutto il resto deve essere gestito da server che operano sopra al microkernel, quindi in spazi di indirizzi separati.
Magari argomentare menzionando anche le politiche e i meccanismi, pro e contro di entrambi, menzionare anche il fatto che nei microkernel viene quindi ridotta la parte di codice eseguita con i massimi privilegi.

1.2 SEL4

seL4 fa parte della famiglia dei microkernel L4 che risalgono alla prima metà degli anni '90 creato da Jochen Liedtke per sopperire alle scarse performance dei primi sistemi operativi basati su microkernel.

2

ESERCIZI

1. Scrivere le possibili evoluzioni del programma

   ```
   co X: = X+2 // X: = X+1 oc
   ```

   assumendo che ciascun assegnamento è realizzato da tre azioni atomiche che caricano X in un registro (`Load R X`), incrementano il valore del registro (`Add R v`) e memorizzano il valore del registro((`Store R X`). Per ciascuna delle esecuzioni risultanti dall'interleaving delle azioni atomiche descrivere il contenuto dopo ogni passo della locazione condivisa X e dei registri privati, R_1 del processo che esegue il primo assegnamento ed R_2 per il processo che esegue il secondo assegnamento. Se assuma che il valore iniziale di X sia 50.

2. Si definisca il problema della *barrier synchronization* e si descrivano per sommi capi i differenti approcci alla sua soluzione. Se ne fornisca quindi una soluzione dettagliata utilizzando i semafori.

3. Considerare n api ed un orso che possono avere accesso ad una tazza di miele inizialmente vuota e con una capacità di k porzioni. L'orso dorme finchè la tazza è piena di k-porzioni, quindi mangia tutto il miele e si rimette a dormire. Le api riforniscono in continuazione la tazza con una porzione di miele finchè non si riempie; l'ape che aggiunge la k-esima porzione sveglia l'orso. Fornire una soluzione al problema modellando orso ed api come processi e utilizzando un monitor per gestire le loro operazioni sulla tazza. Prevedere che le api possano eseguire l'operazione *produce-honey* anche concorrentemente.

4. Descrivere le primitive di scambio messaggi send e receive sia sincrone che asincrone ed implementare

```
- synch_send(v:int)
- send(v:int)
- receive(x:int)
```

utilizzando le primitive di LINDA.

BIBLIOGRAFIA

[1] Autore - *titolo*

[2] Autore - *Titolo* - altre informazioni

[3] Autore - *Titolo3* - altre informazioni