POLITECNICO DI TORINO

# Analisi comparativa tra Os161 e Xv6

Progetto di Programmazione di Sistema - AA 23/24

**PRESENTATO DA:**
Nunzio Messineo
Gabriele Martina

**CONTATTO:**
**s315067@studenti.polito.it**
**s310789@studenti.polito.it**

# Introduzione

## Xv6

Sistema operativo con scopi didattici basato su UNIX V6, progettato dal MIT nel 2006

Realizzato per lo più in C a Assembly

Progettato per essere eseguibile su architetture x86 nella versione del 2006 e RISC-V nel 2020

Architettura in studio: x86

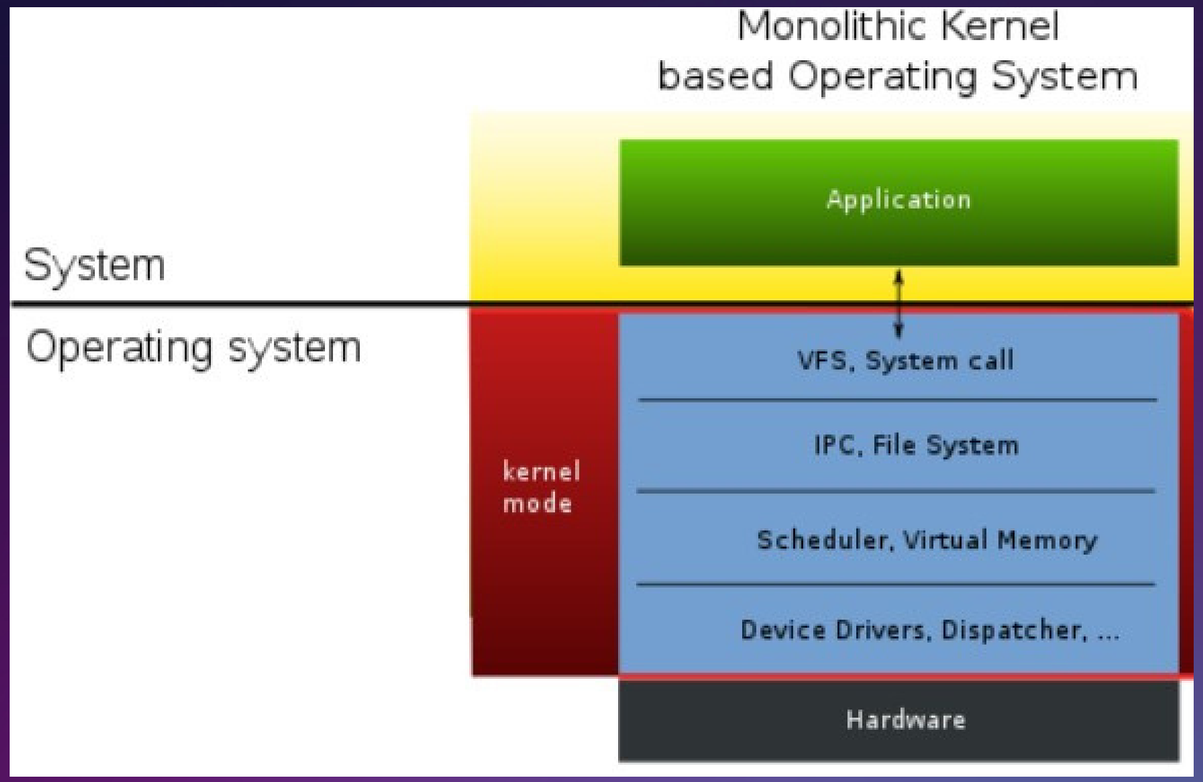
## Os161

Os161 è progettato per l'istruzione e sviluppato presso l'Università di Harvard

Realizzato per lo più in C a Assembly

Supporta diverse architetture, inclusi MIPS, ARM e Intel

Architettura in studio: MIPS

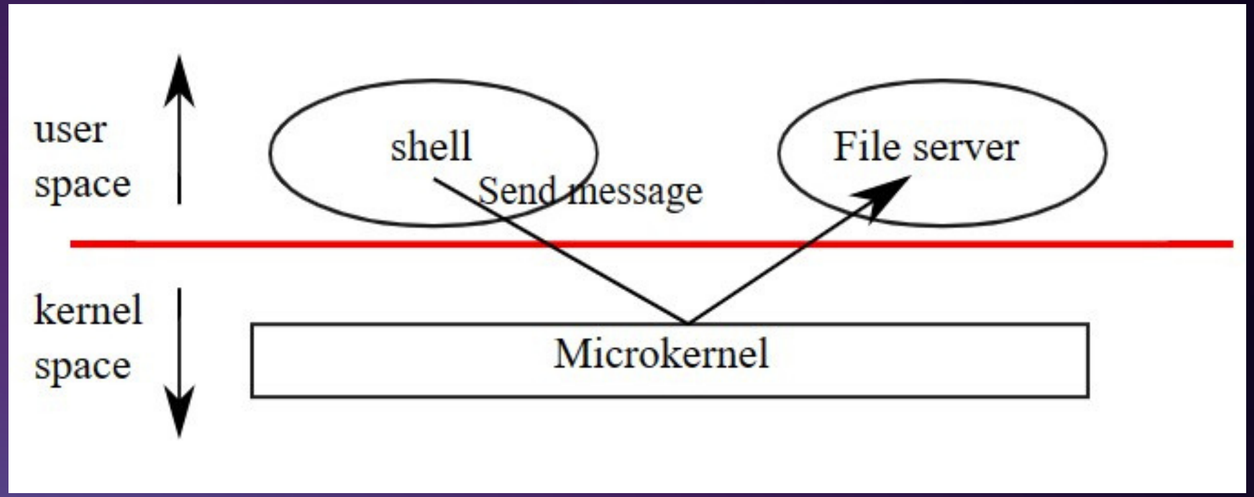



# Struttura del Kernel

Il sistema operativo ha una struttura **monolitica**. L'intero sistema operativo viene eseguito con pieno privilegio hardware. Poiché Xv6 non fornisce molti servizi, il suo kernel è più piccolo di alcuni microkernel.

# Struttura del Kernel di Os161

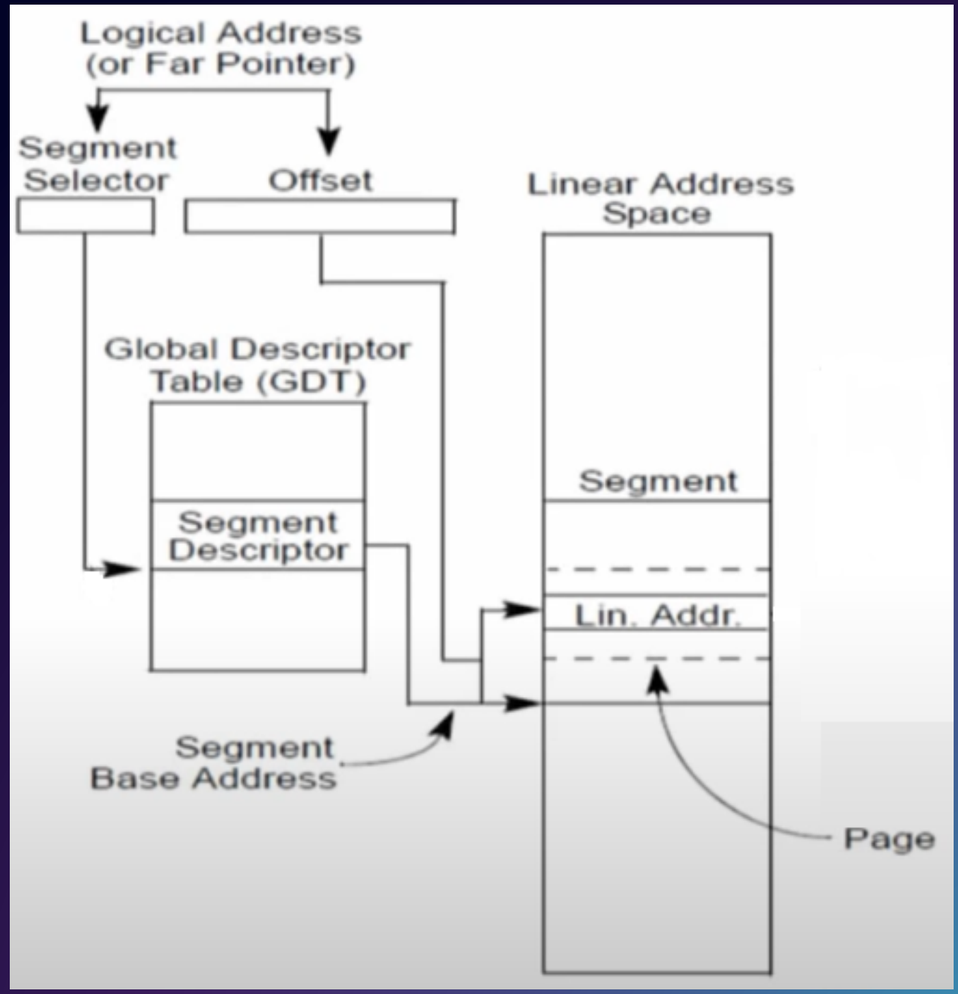
Utilizza un'organizzazione a microkernel, dove solo le funzioni essenziali risiedono nel kernel, mentre i servizi aggiuntivi sono implementati come server a livello utente.

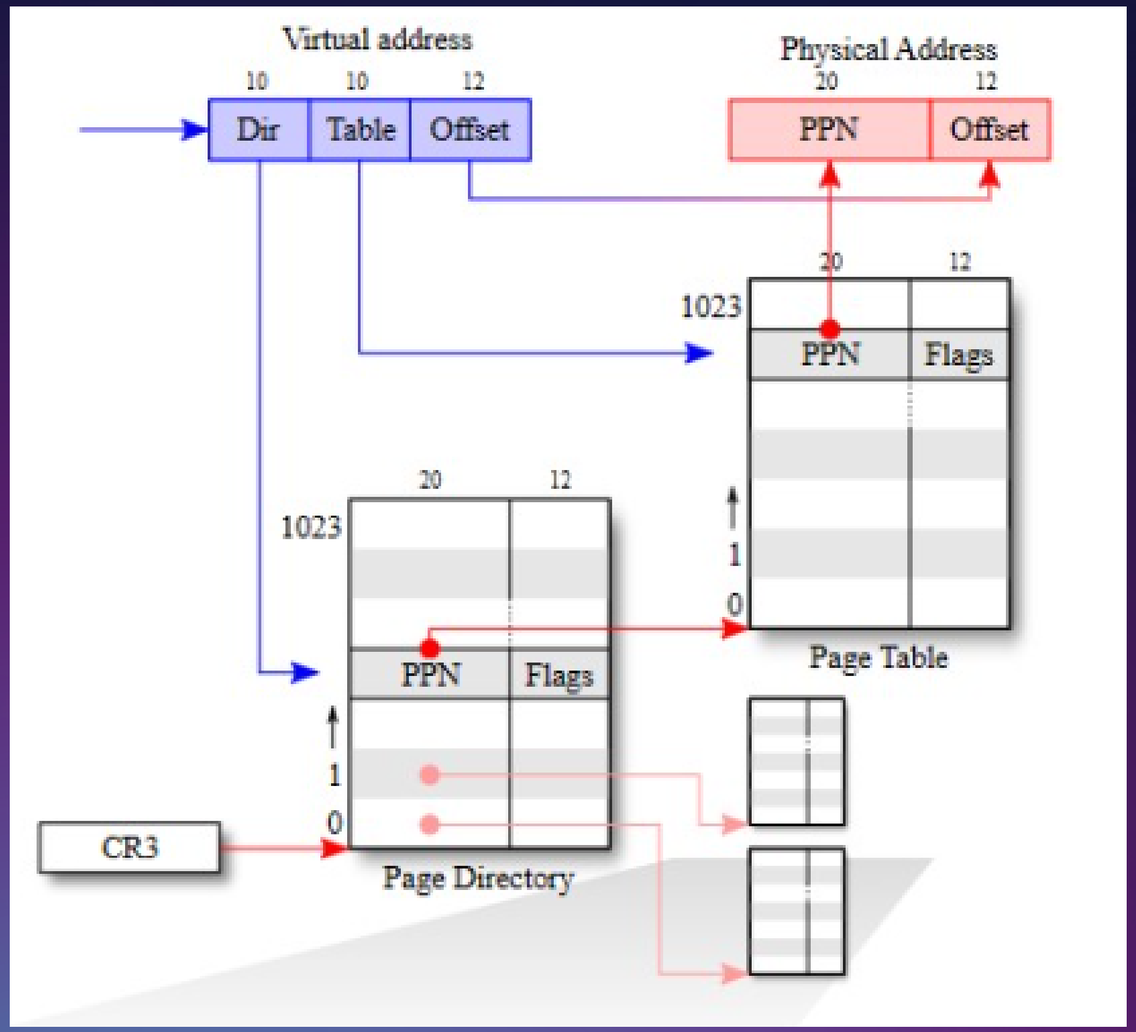


# Gestione della memoria

Implementa un sistema di gestione della memoria virtuale basato su **paging**.
Il paging è diviso in due fasi:

- Fase di **segmentazione** in cui si ottiene 'l'indirizzo lineare' partendo dall'indirizzo virtuale.






# Gestione della memoria

- Le **page tables** consentono il controllo degli indirizzi di memoria, permettendo a Xv6 di multiplexare gli spazi degli indirizzi di processi diversi su una singola memoria fisica e proteggere le memorie di processi diversi.

# Gestione della memoria Os161

## Analogie

Come in xv6 adotta la virtual memory tramite l'utilizzo di page table

## Differenze

Os161 implementa varie tecniche avanzate per ottimizzare l'utilizzo della memoria, tra cui l'utilizzo di memoria condivisa (shared memory), il demand paging e la condivisione in lettura/scrittura (Copy-On-Write, COW).

# System Call, Exceptions e Interrupts

In Xv6 vengono tutte gestite da una stessa funzione. Come prima cosa bisogna effettuare il context switch per passare da user mode a kernel mode.
Nello specifico il sistema operativo deve:

1. Salvare i registri del processore per un futuro ripristino trasparente.
2. Essere configurato per l'esecuzione nel kernel scegliendo un punto in cui far iniziare l'esecuzione del kernel.
3. Il kernel deve essere in grado di recuperare informazioni sull'evento, ad esempio gli argomenti della chiamata di sistema.

# Gestori di trap in Xv6

- La gestione delle trap viene gestita a livello assembly tramite la funzione **int n** (in usys.S) (dove nel caso delle system call n = T_SYSCALL)
- Successivamente avviene la creazione del trap frame con il salvataggio dei registri nella funzione **alltraps**
- int n infine chiama trap a livello c che determina cosa ha scatenato l'interrupt
- Per le system call invoca **syscall** in (syscall.c) che chiama la funzione corrispondente all'ID (in %eax)
- **trapret** (in trapasm.S) esegue il context switch inverso e conclude la chiamata

# Creazione di una system call

L'aggiunta di una nuova system call a Xv6 richiede la modifica del codice del kernel e l'aggiunta di una nuova voce alla system call table:

1. Definire una nuova system call nel codice del kernel
   *#define SYS_getyear 22*
2. Aggiungerla alla system call table (in syscall.h) con un codice univoco
   *[SYS_getyear] sys_getyear*
3. Aggiungere un  puntatore alla chiamata di sistema (in syscall.c solo il prototipo)
   *extern int sys_getyear(void)*
4. Aggiungere l'interfaccia utente in usys.S
   *SYSCALL(getyear)*
5. Implementare la  system call function in sysproc.c
   *int sys_getyear(void) {  return 1975; }*
6. Modificare il Makefile
   *make clean; make*

# Gestione delle trap in Os161

In Os161, allo stesso modo di Xv6, ha un solo handler per gestire system call, exception e interrupt. Le principali differenze sono:

- *Differenze di carattere implementativo*: dovute principalmente all'architettura (MIPS vs x86)
- *Astrazione e modularità*: OS161 promuove una maggiore astrazione e modularità rispetto a xv6. Le system call sono implementate come chiamate remote attraverso messaggi, il che consente una maggiore separazione tra il codice utente e il codice kernel.
- *Isolamento e protezione*: Grazie alla separazione dei servizi e dei componenti nel microkernel di OS161, è più probabile che il sistema operativo offra un maggiore isolamento e protezione tra le diverse componenti. Questo è vantaggioso in termini di sicurezza e stabilità

Anche l'implementazione delle system call hanno sostanzialmente lo stesso funzionamento.

# Sincronizzazione in Xv6

I meccanismi di sincronizzazione implementati da Xv6 sono:

- ***Spin Lock***: è un tipo di lock caratterizzato principalmente da un booleano che determina se è detenuto o è disponibile.
- ***Sleep Lock***: è un tipo di lock che può essere rilasciato e acquisito da un thread diverso da quello che lo detiene.
- ***Wakeup*** e ***Sleep***: Sleep e Wakeup consentono ai processi nello stato di sleeping di dormire in attesa di un evento e ad un altro processo di svegliarlo una volta che l'evento è avvenuto.

# Sincronizzazione in Os161

Os161, allo stesso modo di Xv6, implementa spinlock, mutex lock (sleep lock) e wait channel (wakeup e sleep) però aggiunge una tecnica di sincronizzazione più sofisticata che è quella dei semafori.

Inoltre a differenza di xv6, in Os161 i processi hanno un campo extra che ne definisce la priorità in modo che i thread con priorità più bassa non possono interrompere l'esecuzione di un thread con priorità superiore.

# Scheduling - Xv6

Xv6 implementa un semplice algoritmo di scheduling a **round-robin** che assegna a ciascun processo un intervallo di tempo fisso, chiamato **quantum**. É un algoritmo a priorità fissa in quanto tutti i processi hanno la stessa importanza nell'essere eseguiti.

Il multiplexing dei processi avviene usando il meccanismo di sleep e wakeup. Lo scheduling è costituito principalmente dalle seguenti fasi:

- Acquisizione del lock ptable.lock
- Context switch con l'uso della funzione swtch
- Algoritmo di scheduling
- Ritorno al contesto precedente
- Rilascio del lock ptable.lock

# Scheduling - Os161

Analogamente a Xv6, Os161 implementa di default utilizza una coda unica round-robin.
Come principale differenza in Os161 sono già definite le priorità in modo da assegnare ad ogni thread un’importanza diversa a seconda delle funzionalità o del contesto.

# File System - Xv6

Il file system di Xv6 è suddiviso nei seguenti layer:

- Il livello **Disk**: legge e scrive blocchi su un hard disk IDE.
- Il livello **Buffer cache**: ha due funzioni fondamentali: sincronizzare l'accesso ai blocchi del disco e memorizzare nella cache i blocchi più popolari
- Il livello **Logging**: consente ai livelli superiori di incapsulare gli aggiornamenti a diversi blocchi in una transazione e garantisce che i blocchi vengano aggiornati atomicamente in caso di arresti anomali.
- Il livello **Inode** e **Block Allocator**: fornisce file senza nome, ciascuno rappresentato utilizzando un inode e una sequenza di blocchi che contengono i dati del file.

# File System - Os161

Il file system di Os161 è chiamato Simple File System (SFS) e possiede al suo interno buona parte delle funzionalità di base di un fs:

- Directory gerarchiche
- Supporto per file di grandi dimensioni tramite blocchi indiretti multilivello
- Blocco a grana fine
- Un cache buffer di dati e metadati del file system