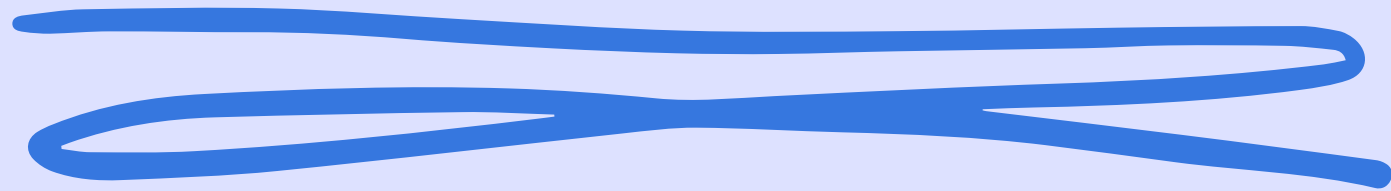




# Analisi comparativa tra OS161 e xv6



**Gruppo 41** Nicolò Marsicano  
Ernesta Maria Sichetti  
Luca Romeo



# ***Indice***

- DIFFERENZE GENERALI
- GESTIONE DELLE SYSCALLS
- MECCANISMI DI SINCRONIZZAZIONE
- MEMORIA VIRTUALE E MEMORY MANAGEMENT UNIT
- ALGORITMI DI SCHEDULING
- INTRODUZIONE CODICE XV6
- GESTIONE DELLE PRIORITA'
- GESTIONE DELLO SCHEDULING

# DIFFERENZE GENERALI TRA OS161 E XV6

 ENTRAMBI PER ESSERE UTILIZZATI IN AMBITO DIDATTICO MA

**OS161 E' PIU' COMPLETO!**

**OS161** E' INFATTI PENSATO PER ASSOMIGLIARE DI PIU' AD  
UN SISTEMA OPERATIVO REALE. CI SONO QUINDI  
DIFFERENZE SOSTANZIALI NELLA GESTIONE DEGLI ELEMENTI  
FONDAMENTALI DI UN SISTEMA OPERATIVO

# IMPLEMENTAZIONE DELLE SYSTEM CALLS

OS161

1. OS161 FORNISCE UNA LIBRERIA DI SUPPORTO UTENTE DEDICATA PER SEMPLIFICARE L'INVOCAZIONE DELLE SYSTEM CALLS
2. LA GESTIONE DEGLI ERRORI PUO' COINVOLGERE I VALORI RESTITUITI DALLE VARIE SYSTEM CALLS
3. SONO PRESENTI TABELLE DEDICATE ALLA GESTIONE DELLE SYSTEM CALLS

1. XV6 GESTISCE MANUALMENTE L'INVOCAZIONE DELLE SYSTEM CALLS NON ESSENDO PRESENTI LIBRERIE DI SUPPORTO
2. IN XV6 GLI ERRORI POSSONO ESSERE GESTITI TRAMITE LA VARIABILE GLOBALE "ERRNO"
3. LE TABELLE NON SONO PRESENTI, VIENE UTILIZZATO UN ARRAY DI PUNTATORE A FUNZIONE CHIAMATO "SYSCALL VECTOR"

XV6

# MECCANISMI DI SINCRONIZZAZIONE

**XV6**

1. **XV6 UTILIZZA LOCK E CONDITION VARIABLES**
2. **I LOCK SONO USATI PER GARANTIRE LA MUTUA ESCLUSIONE**
3. **LE CONDITION VARIABLES SONO USATE PER SINCRONIZZARE I THREAD**

1. **OS161 UTILIZZA LOCK, CONDITION VARIABLES, SEMAFORI E MONITOR**
2. **I SEMAFORI SONO IMPLEMENTATI TRAMITE WAIT CHANNEL E SPINLOCK E GARANTISCONO L'ACCESSO ESCLUSIVO ALLE RISORSE**
3. **I MONITOR FORNISCONO UNA SINCRONIZZAZIONE DI PIÙ ALTO LIVELLO COMBINANDO LOCK E CONDITION VARIABLES**

**OS161**

# MEMORIA VIRTUALE E MMU

**XV6**

1. **XV6 GESTISCE LA MEMORIA VIRTUALE TRAMITE UNA STRUTTURA DI PAGE TABLE SU DUE LIVELLI**
2. **OGNI PROCESSO HA LA PROPRIA PAGE TABLE CHE MAPPA GLI INDIRIZZI VIRTUALI IN INDIRIZZI FISICI, E QUANDO VIENE CREATO UN NUOVO PROCESSO VIENE ALLOCATA UNA NUOVA PAGE TABLE**
3. **LA MMU SI OCCUPA DI TRADURRE GLI INDIRIZZI VIRTUALI IN FISICI**
4. **QUANDO UN PROCESSO ACCEDE A UN INDIRIZZO VIRTUALE, LA MMU CERCA IL CORRISPONDENTE INDIRIZZO FISICO NELLA PAGE TABLE E, SE L'INDIRIZZO NON È NELLA MEMORIA FISICA, AVVIENE UN PAGE FAULT**

1. **IN OS161 POTREBBERO ESSERE MECCANISMI PIÙ AVANZATI CHE GARANTISCONO UNA GESTIONE PIÙ EFFICIENTE DELLA MEMORIA**

**OS161**

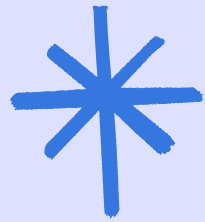
# GESTIONE DELLO SCHEDULING

OS161

1. OS161 IMPLEMENTA IL MULTILEVEL FEEDBACK QUEUE (MLFQ), ASSEGNANDO I PROCESSI A DIVERSE PRIORITY QUEUE IN BASE A QUANTO UTILIZZANO LA CPU
2. OGNI CODA HA IL PROPRIO TIME QUANTUM
3. IL MLFQ SI ADATTA MEGLIO A SITUAZIONI DIVERSE MA AGGIUNGE COMPLESSITÀ

1. XV6 IMPLEMENTA IL ROUND ROBIN, CHE DA A OGNI PROCESSO LO STESSO TEMPO PER L'ESECUZIONE
2. IL ROUND ROBIN È PIÙ SEMPLICE MA LE PERFORMANCE NON SONO OTTIME IN CASO DI CARICHI DI LAVORO VARIABILI

XV6



# **IMPLEMENTAZIONI EFFETTUATE SU XV6**





# INTRODUZIONE DELLE PRIORITA'

ABBIAMO INTRODOTTO UN MECCANISMO DI GESTIONE DELLE PRIORITÀ CHE ASSEGNA UNA PRIORITÀ ( OVVERO UN INTERO COMPRESO TRA 0 E 20 DOVE ZERO RAPPRESENTA IL VALORE PIÙ ALTO ) AD OGNI PROCESSO CREATO DA XV6. LA STRUCT CHE È ADIBITA A CONTENERE LE INFORMAZIONI SU OGNI PROCESSO (“PROC”) È STATA QUINDI MODIFICATA COME SEGUE:


```
// Per-process state
struct proc {
    uint sz;                // Size of process memory (bytes)
    pde_t* pgdir;           // Page table
    char *kstack;           // Bottom of kernel stack for this process
    enum procstate state;   // Process state
    int pid;                // Process ID
    struct proc *parent;    // Parent process
    struct trapframe *tf;   // Trap frame for current syscall
    struct context *context; // switch() here to run process
    void *chan;             // If non-zero, sleeping on chan
    int killed;             // If non-zero, have been killed
    struct file *ofile[NOFILE]; // Open files
    struct inode *cwd;      // Current directory
    char name[16];          // Process name (debugging)
    int priority;
};
```



**LA PRIORITÀ DI OGNI  
PROCESSO È IMPOSTATA A  
10 DI DEFAULT**





# MECCANISMO DI SCHEDULING





PER UTILIZZARE MEGLIO LA NUOVA IMPLEMENTAZIONE DELLE PRIORITÀ ABBIAMO APPORTATO DELLE MODIFICHE ANCHE AL MECCANISMO DI SCHEDULING: LA NUOVA FUNZIONE DI SCHEDULING SI PRESENTA COME SEGUE E PERMETTE DI ASSOCIARE AD OGNI PROCESSO LA MEDESIMA PERCENTUALE DI RISORSE


L'ALGORITMO SELEZIONA TRA I PROCESSI RUNNABLE QUELLO A PIÙ ALTA PRIORITÀ (P1) ED ESEGUE IL CONTEXT SWITCH DELLA CPU, SALVANDONE REGISTRI E CONTESTO, CON IL PROCESSO ATTUALMENTE IN ESECUZIONE (P2), QUINDI P2 PASSA ALLO STATO RUNNABLE MENTRE P1 ENTRA IN ESECUZIONE E DIVENTA RUNNING.




# SYSCALL AGGIUNTIVE



ABBIAMO INTRODOTTO DUE SYSCALL AGGIUNTIVE,  
OGNUNA DELLE QUALI È ADIBITA AD UNO SCOPO BEN  
PRECISO



1. **SYS\_CPS**: permette all'utente di verificare lo stato dei processi creati fino al lancio della syscall, mostrando sul terminale varie informazioni: process identifier, priorità e nome del processo.
  2. **SYS\_CHPR**: fornendo in input il process identifier e un numero relativo alla priorità permette di modificare la priorità di quel determinato processo.
- 

# SCRIPT PER IL TESTING

I TRE SCRIPT CHE ABBIAMO CREATO PER  
VERIFICARE IL CORRETTO FUNZIONAMENTO  
DELLE SYSCALL AGGIUNTE E DELLA NUOVA  
POLICY DI SCHEDULING SONO : PS.C, NICE.C E  
FOO.C

# SCRIPT PER IL TESTING



**PS.C**

ESEGUENDO IL COMANDO PS DAL MENÙ DI XV6 VERRÀ VISUALIZZATA UNA LISTA DEI PROCESSI ATTUALMENTE ATTIVI SUL SISTEMA OPERATIVO INSIEME AD UNA SERIE DI INFORMAZIONI AGGIUNTIVE SUI PROCESSI STESSI.


**NICE.C**

ESEGUENDO IL COMANDO NICE DAL MENÙ DI XV6 È POSSIBILE MODIFICARE LA PRIORITÀ DI CONSEGUENZA LO STATO DEI VARI PROCESSI. IL PRIMO ARGOMENTO INDICA IL PID DEL PROCESSO TARGET MENTRE IL SECONDO LA PRIORITÀ CHE GLI SI VUOLE ASSEGNARE



**FOO.C**

ESEGUENDO IL COMANDO FOO È POSSIBILE CREARE UNA SERIE DI PROCESSI FIGLI PARTENDO DALLO STESSO PROCESSO PADRE. IL PRIMO ARGOMENTO INDICA IL NUMERO DI FIGLI DA GENERARE.



THANK  
YOU