ANALISIS xv6-riscv original

XV6 es un sistema operativo didáctico basado en Unix Version 6, reimplementado para arquitecturas modernas como RISC-V. Desarrollado inicialmente en 2006 por el MIT, sirve como herramienta educativa para comprender los conceptos fundamentales de sistemas operativos. Este informe analiza exhaustivamente sus componentes críticos (gestión de memoria y planificación de procesos) y propone mejoras significativas implementadas en una versión modificada.

Objetivos del informe:

1. Analizar el funcionamiento original de xv6

Entorno de desarrollo:

- Sistema base: Ubuntu 22.04 LTS
- QEMU: versión 7.2 o superior
- Compilador: GCC RISC-V cross-compiler
- Repositorio base: xv6-riscv del MIT (MIT License)

---

2. ANÁLISIS DEL MANEJO DE MEMORIA ORIGINAL

2.1. Arquitectura de Memoria en xv6

XV6 implementa un sistema de memoria virtual con paginación de dos niveles, típico de arquitecturas RISC-V de 64 bits. El espacio de direcciones se divide entre kernel y procesos de usuario.

Estructura del espacio de direcciones:

```
text
0x0000000000000000 - 0x000000000FFFFFFF: User Space (256MB)
0xFFFFFFFF00000000 - 0xFFFFFFFFFFFFFFFF: Kernel Space (4GB)
```

2.2. Tablas de Página

Estructura de PTE (Page Table Entry):

```
c
// Definición en kernel/riscv.h
typedef uint64 pte_t;

// Bits de la PTE (Page Table Entry)
#define PTE_V (1L << 0) // Valid
#define PTE_R (1L << 1) // Read
#define PTE_W (1L << 2) // Write
#define PTE_X (1L << 3) // Execute
#define PTE_U (1L << 4) // User accessible
```

Jerarquía de tablas de página:

- Nivel 1: Página de directorio (4096 entradas)
- Nivel 0: Página de tabla (4096 entradas)
- Tamaño de página: 4KB (4096 bytes)

2.3. Estructuras de Datos Clave

Estructura `proc` (kernel/proc.h):

```
c
struct proc {
  struct spinlock lock;
  pagetable_t pagetable;  // User page table
  uint64 sz;              // Size of process memory (bytes)
  // ... otros campos
};
```

Funciones principales de gestión de memoria:

1. `uvmalloc()` - Asigna páginas físicas y las mapea:

```
c
uint64 uvmalloc(pagetable_t pagetable, uint64 oldsz, uint64 newsz, int
xperm)
```

2. `uvmdealloc()` - Libera páginas:

```c
uint64 uvmdealloc(pagetable_t pagetable, uint64 oldsz, uint64 newsz)
```

3. `walk()` - Busca PTE para dirección virtual:

```c
pte_t *walk(pagetable_t pagetable, uint64 va, int alloc)
```

4. `mappages()` - Crea mapeo VA→PA:

```c
int mappages(pagetable_t pagetable, uint64 va, uint64 size, uint64 pa, int perm)
```

2.4. Esquema de Asignación Original

Alocación de memoria para procesos:

1. `exec()`: Carga programa en memoria
   - Reserva espacio para código, datos, pila
   - Stack fijo de 8KB (2 páginas)
   - Asignación eager: todas las páginas se asignan inmediatamente
2. `sbrk()`: Cambia tamaño del heap
   - Incrementa `p->sz`
   - Llama a `uvmalloc()` para asignar páginas físicas inmediatamente
   - Problema: Asigna toda la memoria solicitada, aunque no se use

Diagrama del flujo de asignación original:

```text
sbrk(n) → sys_sbrk() → growproc() → uvmalloc()
                                        ↓
                        Asignación INMEDIATA de n bytes
                                        ↓
                        Páginas físicas asignadas y mapeadas
```

2.5. Limitaciones del Sistema Original

1. Uso ineficiente de memoria:

    - Procesos que reservan 1GB pero usan 10MB desperdician 990MB
    - No hay asignación bajo demanda (lazy allocation)
2. Stack fijo e ineficiente:
    - 8KB por proceso (2 páginas) siempre asignados
    - No configurable
3. Sin recuperación de recursos huérfanos:
    - Inodos huérfanos permanecen en disco tras crashes

---

3. ANÁLISIS DEL PLANIFICADOR ORIGINAL

3.1. Estructuras del Planificador

Tabla de procesos (kernel/proc.c):

```
c

struct proc proc[NPROC];  // Array estático de NPROC estructuras
```

Estructura `cpu` (kernel/proc.h):

```
c
struct cpu {
  struct proc *proc;      // Proceso actualmente ejecutándose
  struct context context; // Contexto del scheduler
  // ... otros campos

};
```

Contexto de ejecución:

```
c
struct context {
  uint64 ra;
  uint64 sp;
  uint64 s0;
  uint64 s1;
  // ... 12 callee-saved registers

};
```

3.2. Algoritmo de Planificación Original

Política: Round-Robin modificado con selección por PID mínimo

Flujo del scheduler original (`kernel/proc.c`):

```
c
void scheduler(void) {
  struct proc *p;
  struct cpu *c = mycpu();

  c->proc = 0;
  for(;;) {
    // PRIMERA PASADA: Encontrar el menor PID entre procesos RUNNABLE
    int bestpid = 0;
    for(p = proc; p < &proc[NPROC]; p++) {
      acquire(&p->lock);
      if(p->state == RUNNABLE) {
        if(bestpid == 0 || p->pid < bestpid) {
          bestpid = p->pid;
        }
      }
      release(&p->lock);
    }

    if(bestpid == 0)
      continue;  // Nada que ejecutar

    // SEGUNDA PASADA: Localizar el proceso con bestpid
    for(p = proc; p < &proc[NPROC]; p++) {
      acquire(&p->lock);
      if(p->pid == bestpid && p->state == RUNNABLE) {
        // Cambiar al proceso
        p->state = RUNNING;
        c->proc = p;
        swtch(&c->context, &p->context);
        c->proc = 0;
        release(&p->lock);
        break;
      }
      release(&p->lock);
    }
  }

}
```

3.3. Funciones Clave del Scheduler

1. `scheduler()` - Loop principal del planificador
2. `sched()` - Cambia al scheduler
3. `yield()` - Cede la CPU
4. `sleep()` - Pone proceso en estado SLEEPING
5. `wakeup()` - Despierta procesos

3.4. Mecanismo de Interrupciones de Timer

Configuración original (kernel/start.c):

```
c
void timerinit() {
  // Timer en M-mode (Machine mode)
  w_mtvec((uint64)timervec);  // Handler en M-mode
  w_mstatus(r_mstatus() | MSTATUS_MIE);
  w_mie(r_mie() | MIE_MTIE);  // Habilitar timer interrupt de machine mode
}
```

Flujo de interrupciones:

```
text
Timer interrupt → M-mode handler (timervec)
                → Software interrupt a S-mode
                → devintr() en S-mode
                → clockintr()
```

3.5. Limitaciones del Planificador Original

1. Ineficiencia en búsqueda: Dos pasadas por la tabla de procesos
2. Sin ahorro de energía: Busy-wait cuando no hay procesos
3. Overhead de timer: Dos niveles de indirección (M-mode → S-mode)
4. Salvado excesivo de registros: 12 registros callee-saved innecesarios

---