Trabalho Prático 2 (TP2) - Sistemas Operacionais com xv6

Introdução

Este repositório contém a implementação do Trabalho Prático 2 (TP2) para o sistema operacional xv6. O objetivo principal é explorar o gerenciamento de memória e a criação de chamadas de sistema, com ênfase na implementação do conceito de Copy-on-Write (COW) para otimização de memória.

Objetivos Gerais

- Adicionar chamadas de sistema personalizadas.
- Explorar o gerenciamento de tabelas de páginas em arquiteturas x86.
- Implementar o conceito de Copy-on-Write (COW).

Autores

- **Marcos Lorran B. Aguilar Aguilar**
- **Gustavo Dias Apolinario**

Estrutura do Trabalho

TP2.1: Implementação da chamada `date`

- **Objetivo:** Criar uma chamada de sistema que exibe a data e hora atual.
- **Arquivos principais modificados:**
- `syscall.c`, `syscall.h`: Definição e registro da chamada.
- `user.h`: Cabeçalho da chamada de sistema.
- `sysproc.c`: Implementação da lógica da chamada de sistema.

TP2.2: Implementação de chamadas auxiliares

Função `virt2real`

- **Descrição:** Converte um endereço virtual para um endereço físico.
- **Protótipo:**

```
char* virt2real(char *va);
```

- **Funcionamento:** Usa a função `walkpgdir` para acessar a tabela de páginas e retornar o endereço físico correspondente.

Função `num_pages`

- **Descrição:** Retorna o número de páginas alocadas pelo processo atual.
- **Protótipo:**

```
int num_pages(void);
```

- **Funcionamento:** Conta as páginas referenciadas no espaço de memória do processo atual.

TP2.3: Implementação do Copy-on-Write (COW)

- **Objetivo:** Criar uma chamada de sistema `forkcow`, similar à `fork`, mas com suporte a Copy-on-Write.
- **Diferenciais:**
- As páginas do processo pai e do filho são inicialmente compartilhadas como Read-Only.
- Na tentativa de escrita, é gerada uma cópia exclusiva da página.

Passos para Implementação

1. **Gerenciar Referências de Páginas:** Criar um contador para rastrear o número de processos que compartilham cada página.
2. **Marcar Páginas como Read-Only:** Alterar permissões na tabela de páginas.
3. **Detectar Page Faults:** Tratar falhas de página geradas por escrita.
4. **Criar Páginas Exclusivas:** Alocar novos frames de memória conforme necessário.
5. **Flush na TLB:** Garantir consistência entre a tabela de páginas e o cache de TLB após alterações.

Arquivos Modificados

user.h

- Funções adicionadas:

```
int date(void*);
char* virt2real(char *va);
int num_pages(void);
int forkcow(void);
```

- Define as chamadas visíveis ao usuário ("biblioteca padrão" do xv6).

syscall.h

- Adicionadas definições para as novas chamadas de sistema:

```
#define SYS_date 22
#define SYS_virt2real 23
#define SYS_num_pages 24
#define SYS_forkcow 25
```

syscall.c

- Registro das novas chamadas:

```
extern int sys_virt2real(void);
extern int sys_num_pages(void);
extern int sys_forkcow(void);
[SYS_virt2real] sys_virt2real,
[SYS_num_pages] sys_num_pages,
[SYS_forkcow] sys_forkcow,
```

usys.S

- Adicionadas macros para chamadas de sistema:

```
SYSCALL(date)
SYSCALL(virt2real)
SYSCALL(num_pages)
SYSCALL(forkcow)
```

sysproc.c

- Implementação das chamadas:

Exemplo: `sys_forkcow`

```
int sys_forkcow(void) {
return forkcow();
}
```

Exemplo: `sys_date`

```
int sys_date(void) {
char *ptr;
argptr(0, &ptr, sizeof(struct rtcdate*));
struct rtcdate* r = (struct rtcdate*)ptr;
cmostime(r);
return 0;
}
```

kalloc.c

- Gerenciamento de referências:

```
int get_ref_count(uint pa) {
int index = pa / PGSIZE;
if (index < PHYSTOP / PGSIZE) {
return ref_counts[index];
}
return -1;
}
void incr_ref_count(uint pa) {
acquire(&kmem.lock);
int index = pa / PGSIZE;
if (index < PHYSTOP / PGSIZE) {
ref_counts[index]++;
}
release(&kmem.lock);
}
void decrement_ref_count(uint pa) {
acquire(&kmem.lock);
int index = pa / PGSIZE;
if (ref_counts[index] > 0) {
ref_counts[index]--;
}
release(&kmem.lock);
}
```

mmu.h

- Adicionado bit para Copy-on-Write:

```
#define PTE_COW 0x200
```

proc.c

- Implementação do `forkcow`:

```
int forkcow(void) {
int i, pid;
struct proc *np;
struct proc *curproc = myproc();
if ((np = allocproc()) == 0) {
return -1;
}
if ((np->pgdir = copyuvmcow(curproc->pgdir, curproc->sz)) == 0) {
kfree(np->kstack);
np->kstack = 0;
np->state = UNUSED;
return -1;
}
np->sz = curproc->sz;
np->parent = curproc;
*np->tf = *curproc->tf;
np->tf->eax = 0;
for (i = 0; i < NOFILE; i++)
if (curproc->ofile[i])
np->ofile[i] = filedup(curproc->ofile[i]);
np->cwd = idup(curproc->cwd);
pid = np->pid;
acquire(&ptable.lock);
np->state = RUNNABLE;
release(&ptable.lock);
return pid;
}
```