Trabalho 3 - Memória Mapeada

June 6, 2017

Memória virtual via paginação

- Cada processo tem sua própria tabela de páginas

```
//processo A
int* p = (int*)0x0004d780;
*p = 665;

assert(*p == 665);
```

```
//processo B
int* p = (int*)0x0004d780;

++*((int*)0x0004d780);
```

Memória compartilhada

- E se dois processos quiserem compartilhar algumas páginas?

Memória compartilhada

- E se dois processos quiserem compartilhar algumas páginas?
- Mapeiam-se a páginas lógicas para a mesma página física

```
//processo A
mmap(0x0004d780, 4096, ...);
int* p = (int*)0x0004d780;
*p = 665;

assert(*p == 666);
```

```
//processo B
mmap(0x0004d780, 4096, ...);
int* p = (int*)0x0004d780;

++*((int*)0x0004d780);
```

Mapeamento de arquivo em memória

- Uma página pode estar na área de swap no disco
- Area de swap pode ser um arquivo

Mapeamento de arquivo em memória

- Uma página pode estar na área de swap no disco
- Area de swap pode ser um arquivo
- É possível mapear uma região da memória para um arquivo
- Multiplos processos podem fazer o mapeamento

Vantages do Mapeamento de arquivo em memória

- Evita cópias:
  - `fread` e `fwrite` envolvem cópias para areas de buffer antes da operação ser efetivada
  - Sockets e (named) pipes também envolvem cópias
- Escrita rápida direto na RAM (sem syscall)
- Cache em RAM grande, tamanho gerenciado pelo SO

Quando usar Mapeamento de arquivo em memória

- Arquivos grandes
- Compartilhados entre processos
- Acessos de tamanhos próximos do tamanho da página
- Vários acessos ao longo do tempo

Quando usar Mapeamento de arquivo em memória

- Arquivos grandes
- Compartilhados entre processos
- Acessos de tamanhos próximos do tamanho da página
- Vários acessos ao longo do tempo
- Cenários ruins:
  - Arquivos pequenos
  - Acessos esparsos de poucos bytes
  - Poucos acessos

mmap

```
FILE* file = fopen(filename, "a+");
```

- Abrir um arquivo

mmap

```
FILE* file = fopen(filename, "a+");
fseek(file, 0, SEEK_END);
size_t len = ftell(file);
```

- Descobrir o tamanho

mmap

```
FILE* file = fopen(filename, "a+");
fseek(file, 0, SEEK_END);
size_t len = ftell(file);
void* base = mmap(NULL, ...);
```

- Criar o mapeamento
- O SO escolhe o endereço virtual

mmap

```
FILE* file = fopen(filename, "a+");
fseek(file, 0, SEEK_END);
size_t len = ftell(file);
void* base = mmap(NULL, len, ...);
```

- Criar o mapeamento
- Tamanho do mapeamento

mmap

```
FILE* file = fopen(filename, "a+");
fseek(file, 0, SEEK_END);
size_t len = ftell(file);
void* base = mmap(NULL, len, PROT_READ|PROT_WRITE, ...);
```

- Criar o mapeamento
- Permissão de leitura e escrita

mmap

```
FILE* file = fopen(filename, "a+");
fseek(file, 0, SEEK_END);
size_t len = ftell(file);
void* base = mmap(NULL, len, PROT_READ|PROT_WRITE, MAP_SHARE
                  ...);
```

- Criar o mapeamento
- Modificações visíveis por outros processos

mmap

```
FILE* file = fopen(filename, "a+");
fseek(file, 0, SEEK_END);
size_t len = ftell(file);
void* base = mmap(NULL, len, PROT_READ|PROT_WRITE, MAP_SHARE
                          fileno(file), 0);
```

- Criar o mapeamento
- Descritor de arquivo mapeado e offset

Em Java

- GC move objetos na memória

Em Java

- GC move objetos na memória
- Solução: `MappedByteBuffer`

MappedByteBuffer

```
FileChannel ch = FileChannel.open(path, READ, WRITE);
```

- java.nio
- Leitura e escrita

MappedByteBuffer

```
FileChannel ch = FileChannel.open(path, READ, WRITE);
MappedByteBuffer mb = ch.map(READ_WRITE, 0, size);
```

- Modo: `READ_WRITE`, `READ_ONLY`, `PRIVATE`
- Offset dentro do arquivo
- Tamanho do mapeamento

MappedByteBuffer

```
FileChannel ch = FileChannel.open(path, READ, WRITE);
MappedByteBuffer mb = ch.map(READ_WRITE, 0, size);
mb.put((byte)127);
mb.position(666);
int value = mb.getInt();
```

- Escreve byte na posição atual

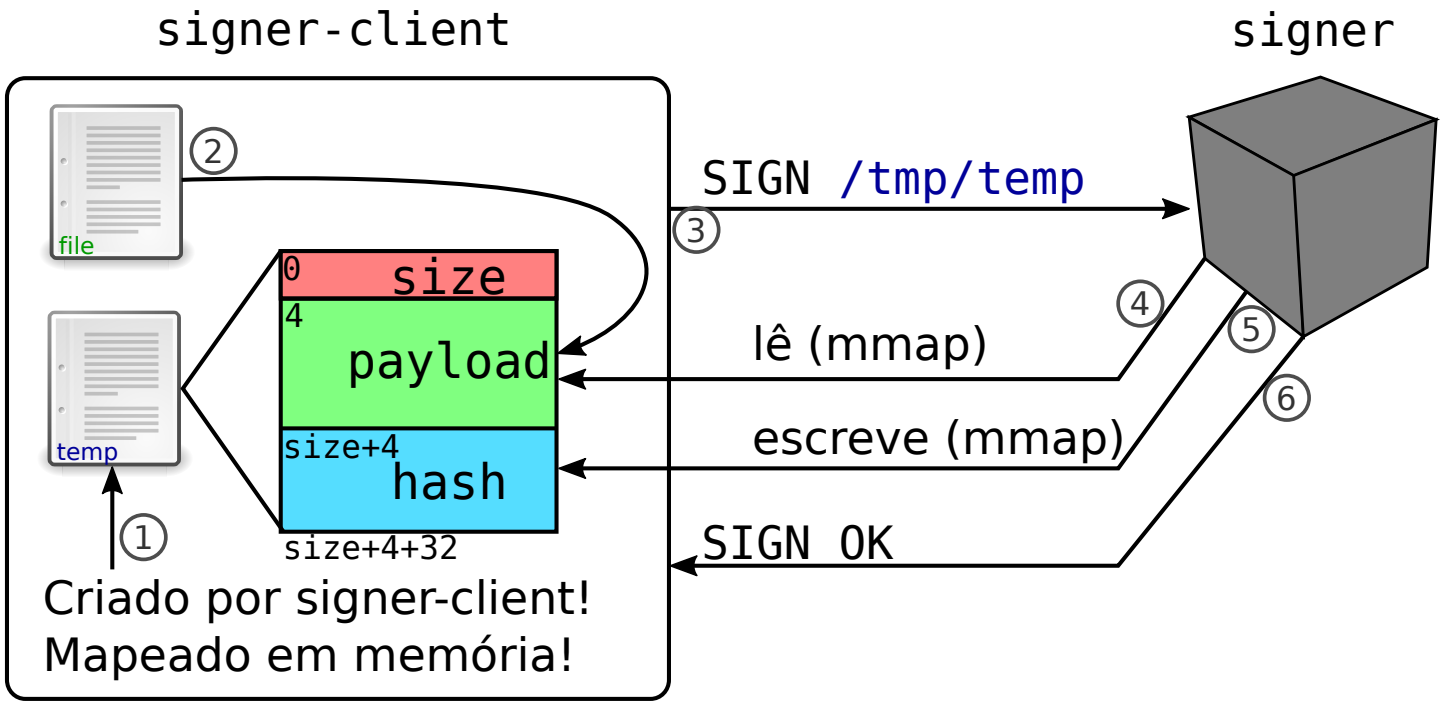
MappedByteBuffer

```
FileChannel ch = FileChannel.open(path, READ, WRITE);
MappedByteBuffer mb = ch.map(READ_WRITE, 0, size);
mb.put((byte)127);
mb.position(666);
int value = mb.getInt();
```

- Escreve byte na posição atual
- Lê um inteiro (4 bytes) da posição 666

Trabalho 3

```
java -jar signer-client-1.0-SNAPSHOT.jar \
                                  /path/to/signer file
```



Pipes

- `signer` recebe comandos via *stdin*
- `echo -ne "SIGN file\nEND " | ./signer`
- Como que o processo pai (`signer-client`) pode mandar comandos?

Pipes

- `signer` recebe comandos via *stdin*
- `echo -ne "SIGN file\nEND " | ./signer`
- Como que o processo pai (`signer-client`) pode mandar comandos?
- Da mesma forma, com um *pipe*

Pipe para stdin/stdout

- Pai cria dois pipes: `pin` e `pou` `pipe(pin);`

Pipe para stdin/stdout

- Pai cria dois pipes: `pin` e `pou`
- Pai chama `fork()`

```
pipe(pin);
```

Pipe para stdin/stdout

- Pai cria dois pipes: `pin` e `pou`
- Pai chama `fork()`

```
pipe(pin);
```

Pai

- Fecha ponta de escr. de `pou` e de leit. de `pin`

Filho

- Fecha ponta de leit. de `pou` e de escr. de `pin`

Pipe para stdin/stdout

- Pai cria dois pipes: `pin` e `pou`
- Pai chama `fork()`

```
pipe(pin);
```

Pai

- Fecha ponta de escr. de `pou` e de leit. de `pin`

Filho

- Fecha ponta de leit. de `pou` e de escr. de `pin`
- Fecha stdout e redireciona pra ponta de escr. de `pou`
- Fecha stdin e redireciona pra ponta de leit. de `pin`

```
dup2(1, pou[1]); dup2(0, pin[0]);
```

Pipe para stdin/stdout

- Pai cria dois pipes: `pin` e `pou`
- Pai chama `fork()`

```
pipe(pin);
```

Pai

- Fecha ponta de escr. de `pou` e de leit. de `pin`
- Lê e escreve das pontas abertas como arquivos

Filho

- Fecha ponta de leit. de `pou` e de escr. de `pin`
- Fecha stdout e redireciona pra ponta de escr. de `pou`
- Fecha stdin e redireciona pra ponta de leit. de `pin`

Pipe para stdin/stdout

- Pai cria dois pipes: `pin` e `pou`
- Pai chama `fork()`

```
pipe(pin);
```

Pai

- Fecha ponta de escr. de `pou` e de leit. de `pin`
- Lê e escreve das pontas abertas como arquivos

Filho

- Fecha ponta de leit. de `pou` e de escr. de `pin`
- Fecha stdout e redireciona pra ponta de escr. de `pou`
- Fecha stdin e redireciona pra ponta de leit. de `pin`
- `exec()`
- Usa stdin e stdout normalmente

Processos em Java

```
ProcessBuilder builder = new ProcessBuilder()
```

- Idioma Builder: parâmetros nomeados

Processos em Java

```
ProcessBuilder builder = new ProcessBuilder()
    .command("/usr/bin/echo", "Quack")
```

- Programa e argumentos

Processos em Java

```
ProcessBuilder builder = new ProcessBuilder()
    .command("/usr/bin/echo", "Quack");
Process process = builder.start();
int exitCode = process.waitFor();
```

- Inicia o processo e espera pelo término

Processos em Java

```
ProcessBuilder builder = new ProcessBuilder()
    .command("/usr/bin/echo", "Quack");
Process process = builder.start();
int exitCode = process.waitFor();
```

- ProcessBuilder já implementa o *piping* dos streams do filho.